N° d'enregistrement C.N.R.S.
A.O 8066

THESE

DE

DOCTORAT D'ETAT ES-SCIENCES MATHEMATIQUES

présentée

A L'UNIVERSITÉ PARIS VI

par

Sacha KRAKOWIAK

pour obtenir

le grade de Docteur Es-Sciences

CONCEPTION ET RÉALISATION DE SYSTEMES A ACCES MULTIPLE : ALLOCATION DES RESSOURCES

Soutenue le 13 février 1973 devant la Commission d'examen

MM. J. ARSAC — *Président*
C. PAIR
P. FEAUTRIER — *Examinateurs*
H. BOUCHER

N° d'enregistrement C.N.R.S.
A.O 8066

THESE

DE

DOCTORAT D'ETAT ES-SCIENCES MATHEMATIQUES

présentée

A L'UNIVERSITÉ PARIS VI

par

Sacha KRAKOWIAK

pour obtenir

le grade de Docteur Es-Sciences

CONCEPTION ET RÉALISATION DE SYSTEMES A ACCES MULTIPLE : ALLOCATION DES RESSOURCES

Soutenue le 13 février 1973 devant la Commission d'examen

MM. J. ARSAC *Président*
C. PAIR
P. FEAUTRIER } *Examinateurs*
H. BOUCHER

UNIV.
LEIDEN
BIBL.

CONCEPTION ET REALISATION

DE SYSTEMES A ACCES MULTIPLE

ALLOCATION DE RESSOURCES

S. KRAKOWIAK

Je tiens à remercier :

Monsieur le Professeur ARSAC, Directeur de l'Institut de Programmation, qui m'a fait l'honneur d'accepter de présider le jury de thèse,

Monsieur le Professeur PAIR, qui a bien voulu se charger du rôle de Rapporteur,

Monsieur le Professeur FEAUTRIER, qui a bien voulu accepter de participer au jury,

Monsieur l'Ingénieur en Chef BOUCHER, qui a défini le projet dans le cadre duquel le présent travail a été effectué.

Je tiens également à remercier tous mes collègues de l'IRIA dont les remarques, les critiques, et la participation active au projet m'ont aidé à réaliser ce travail.

Enfin, j'exprime une reconnaissance particulière à Madame LETAILLIEUR et à Madame PEREZ qui ont assuré avec soin et patience la réalisation matérielle de cet ouvrage.

Digitized by the Internet Archive
in 2026

https://archive.org/details/IA41542715_0017

Sommaire

Introduction

Première Partie

1. Concepts fondamentaux des systèmes à partage de ressources

   1.1 Description des activités parallèles

   1.2 Processus. Machines virtuelles

2. Modèles de comportement des programmes et des usagers

   2.1 Comportement intrinsèque des programmes

   2.2 Comportement des programmes en mémoire paginée : résultats expérimentaux

      2.21 Expériences de SDC

      2.22 Expériences de Princeton

      2.23 Expériences d'IBM en mémoire restreinte

   2.3 Comportement des programmes en mémoire paginée : modèles théoriques

      2.31 Modèle du "working set"

      2.32 Modèles statistiques

         2.321 Modèle à probabilités constantes

         2.322 Modèle markovien stationnaire

         2.323 Modèle à caractéristiques variables

   2.4 Comportement des usagers conversationnels

3. Algorithmes dynamiques d'allocation

   3.1 Le phénomène d'écroulement ("thrashing")

   3.2 Prévention de l'écroulement

      3.21 Allocation statique

      3.22 Allocation dynamique

Deuxième Partie

4. Objectifs et description générale du système ESOPE

4.1 Objectifs et spécifications générales du projet

4.2 Caractéristiques du matériel

4.3 Principes de conception et de réalisation

4.31 Les notions fondamentales

4.32 Techniques de réalisation

5. Gestion des processus et accès à l'information

5.1 Gestion des processus

5.11 Mécanismes de synchronisation entre processus

5.12 Sémaphores

5.121 Définition et propriétés fondamentales

5.122 Utilisation des sémaphores

5.123 Un exemple de synchronisation par sémaphores

5.13 Implantation des processus dans le système ESOPE

5.131 Classes de processus et de sémaphores

5.132 Structures de données associées aux processus

5.133 Implantation des primitives de synchronisation

5.2 Accès à l'information

5.21 Les objectifs et les contraintes

5.22 Choix d'un mécanisme

5.23 Le couplage : structures de données et algorithmes

5.3 Usagers et machines virtuelles

5.31 Choix d'une solution

5.32 Propriétés des usagers

5.33 Opérations sur les fichiers

5.4 Mécanismes de protection dans le système ESOPE

5.41 Introduction

5.42 Mécanismes de protection du 10070

5.421 Restriction des actions

5.422 Restriction des accès à l'information

5.43 Pouvoirs des processus

5.44 Protection de l'information

5.441 Protection interne d'un usager

5.442 Protection de la mémoire physique

6. Conception et réalisation de l'allocateur

6.1 Principes de l'allocation de ressources

6.2 Le cycle des usagers

6.21 Etats des usagers

6.22 Transition entre états

6.23 Etats et files d'usagers

6.3 Allocation d'unité centrale

6.4 Allocation de mémoire

6.41 Algorithme de sélection des usagers courants

6.42 Gestion de la mémoire physique

6.421 Organisation générale

6.422 Structures de données

6.43 Allocation des pages physiques

6.431 Traitement du défaut de page

6.432 Préparation de la requête

6.44 Transferts d'information

6.441 Exécution des requêtes

6.442 Acquittement des requêtes

6.45 Récapitulation

7. Organisation du système en processus parallèles

7.1 Principes de la décomposition

7.2 Mécanismes d'allocation du processeur

7.3 Blocage technologique des prccessus

7.4 Processus de l'allocateur

7.41 Changement d'usager élu

7.42 Exécution des transferts

7.43 Acquittement des requêtes

7.44 Admission et départ des usagers

7.441 Admission

7.442 Départ

7.45 Processus premier des usagers

Troisième Partie

8. Modèle et résultats de simulation du système ESOPE

8.1 Objectifs de la simulation

8.2 Modèle de simulation

8.21 Caractéristiques générales

8.22 Simulation du moniteur

8.23 Simulation de la charge du système

8.3 Exploitation du modèle

8.31 Remarques générales

8.32 Utilisation du modèle

8.4 Gestion de la mémoire physique

8.41 Influence du comportement des programmes

8.42 Influence de la politique de réservation

8.43 Analyse du phénomène d'écroulement

8.5 Echanges de pages

8.51 Etude du préchargement

8.52 Etude du prévidage

8.6 Partage de l'information

8.7 Conclusion sur la simulation

Conclusion

Bibliographie

Annexe

Introduction

Ils n'ont pas tant pensé nous establir quelque certitude, que nous montrer jusques où ils estoyent allez en cette chasse de la vérité.

Montaigne
Essais, I, XL

Introduction

Le présent travail est consacré à l'étude de quelques aspects théoriques et expérimentaux de la conception et de la réalisation des systèmes informatiques à accès multiple. Il a été réalisé à l'Institut de Recherche d'Informatique et d'Automatique, dans le cadre du projet ESOPE (Exploitation Simultanée d'un Ordinateur et de ses PEriphériques), entre 1968 et 1971.

Le projet ESOPE a été mené par une équipe composée de C. BETOURNE, J. FERRIE, C. KAISER, J. MOSSIERE et nous-mêmes. Le présent travail développe particulièrement les problèmes d'allocation de ressources ; les problèmes de gestion de processus parallèles sont traités plus en détail dans la thèse de C. KAISER [K3].

Nous avons choisi de centrer notre étude sur la conception et la réalisation d'un système opérationnel, destiné à un usage effectif, quoique dégagé de toute considération commerciale. Il apparaît en effet (cf. en particulier [S4,] [S5])* qu'une préoccupation majeure des informaticiens est le caractère encore largement empirique des techniques de conception et d'implantation de programmes complexes, et tout particulièrement de ceux où se posent en même temps les problèmes de la gestion d'activités parallèles et de l'attribution de ressources à des demandeurs concurrents, dans une situation de pénurie. Nous pensons donc qu'une présentation logique de la succession de choix, décisions et compromis que constitue l'élaboration du projet d'un système réel, et une évaluation critique du résultat de ces décisions, peuvent apporter une contribution à l'évolution de l'Informatique vers une Science. C'est dans cet esprit qu'a été entrepris le travail ici présenté.

La classe des systèmes informatiques auxquels peut s'appliquer cette étude est celle des systèmes interactifs non spécialisés à accès multiple. Ces systèmes peuvent être caractérisés par les propriétés suivantes :

- ils peuvent servir simultanément un certain nombre d'usagers, chacun d'eux ayant la possibilité d'intervenir à tout instant dans le déroulement de son programme.
- ils ne sont pas limités à un type d'utilisation, mais permettent l'emploi de divers traducteurs et programmes de service, dont la liste est d'ailleurs extensible.
- ils possèdent un système de fichiers permettant la conservation permanente (éventuellement le partage) d'information appartenant aux usagers.

---

* Les notations entre crochets renvoient à la bibliographie *in fine*

Un représentant notoire de cette classe de systèmes est MULTICS, développé au MIT depuis 1964 dans le cadre du projet MAC. Nombre de concepts maintenant classiques sont apparus pour la première fois dans ce projet.

En restreignant l'une ou l'autre des propriétés ci-dessus énumérées, on trouve d'autres classes de systèmes, auxquelles peuvent s'appliquer, *mutatis mutandis*, certaines des idées développées dans cette étude. Citons ainsi, à titre d'exemple, les systèmes spécialisés à accès multiple (du type réservation de places, consultation de fichiers), où la restriction porte sur les programmes exécutés, et les systèmes d'"entrée à distance"où la restriction porte sur les possibilités d'interaction.

Cette étude est divisée en trois parties :

1 - Problèmes fondamentaux de l'allocation de ressources
2 - Allocation de ressources dans le système ESOPE
3 - Résultats de simulations sur le système ESOPE

Plan de la Première Partie

- Le chapitre 1 est une introduction aux concepts fondamentaux utilisés dans les systèmes à partage de ressources. Sont en particulier introduites les notions de processus et de machine virtuelle, ainsi que les états des processus.

- Le chapitre 2 est consacré à l'étude du comportement des usagers d'un système considérés comme des générateurs de demandes de ressources. On s'intéresse au comportement dynamique des programmes dans leur espace de travail (références à la mémoire) et au comportement des usagers conversationnels devant leur console (demandes de processeur). Les résultats des expériences permettent d'établir des modèles utilisés pour l'élaboration de stratégies d'allocation.

- Le chapitre 3 présente, en s'appuyant sur les résultats qui précèdent, une étude des diverses stratégies d'allocation qui visent à traiter de façon globale l'ensemble de ressources constitué par les processeur, la mémoire centrale et la mémoire secondaire.
A l'issue de cette première partie, on dispose des concepts et des méthodes qui seront appliqués dans l'élaboration du système ESOPE.

Plan de la Seconde Partie :

- Le chapitre 4 est une présentation générale du système ESOPE : matériel, spécifications d'ensemble, objectifs et choix fondamentaux, méthodes

de conception et d'implantation.

- Dans le chapitre 5 sont exposés les méthodes utilisées dans ESOPE pour la gestion des processus et pour l'accès à l'information, préliminaires nécessaires à l'étude des mécanismes de l'allocateur proprement dit.

On rappelle le principe du mécanisme de synchronisation par sémaphores, qui est illustré par un exemple tiré du système ESOPE et on donne un aperçu de l'implantation des processus et des primitives de synchronisation.

En ce qui concerne l'accès à l'information, on développe en détail les raisons ayant conduit au choix du mécanisme du "couplage" compte tenu des objectifs recherchés et des contraintes d'implantation. Le choix de la notion d'usager, conduisant à associer une mémoire virtuelle distincte à chaque famille de processus indépendants, est également explicité.

Enfin, le chapitre 5 se termine par un examen des différents problèmes de protection et des solutions qui leur ont été apportées.

- Le chapitre 6 développe les principes de conception de l'allocateur de ressources d'ESOPE, et les mécanismes permettant de mettre en oeuvre ces principes. On montre comment l'application des principes de conception conduit à représenter l'évolution des usagers et du système lui-même, suivant un diagramme d'états. Les algorithmes de gestion des diverses ressources partageables sont ensuite présentés.

- Dans le chapitre 7, prolongement du chapitre précédent, on montre comment les mécanismes de l'allocateur sont implantés sous la forme d'une famille de processus parallèles, dont on décrit la coopération. On montre en particulier comment la notion d'interruption se traduit sous la forme d'activation de processus, permettant une description plus claire du système, rendant plus aisée sa réalisation et sa mise au point.

Plan de la Troisième Partie

- Dans le chapitre 8 sont présentés les résultats obtenus à l'aide d'un modèle de simulation du système ESOPE, réalisé parallèlement au système lui-même. Ce modèle est utilisé pour une évaluation quantitative des stratégies de gestion des ressources centrales du système. On présente en particulier des expériences sur la limitation dynamique du degré de multiprogrammation, qui permet d'éviter le phénomène d'instabilité connu sous le nom d'"écroulement". On évalue également les diverses stratégies permettant de réduire les échanges

entre niveaux de mémoire ; on compare en particulier leur efficacité pour divers types de comportement de programmes.

En conclusion, on tente de tirer quelques enseignements de l'expérience constituée par la conception et la réalisation du système ESOPE ; on examine en particulier les points suivants :

- vérification des hypothèses initiales
- validité des choix fondamentaux de conception
- mise en oeuvre et description des algorithmes.

Enfin, on indique brièvement quelques voies ouvertes aux recherches futures.

Première Partie

Chapitre 1 - Concepts fondamentaux des systèmes à partage de ressources

chapitre 2 - Modèles de comportement des programmes et des usagers

chapitre 3 - Algorithmes dynamique d'allocation.

What does make design a problem in real world cases is that we are trying to make a diagram for forces whose field we do not understand.

Christopher Alexander
Notes on the Design of Form

Chapitre 1

CONCEPTS FONDAMENTAUX DES SYSTEMES A PARTAGE DE RESSOURCES

Un grand nombre de systèmes informatiques modernes sont fondés sur la notion de partage. A la base de la conception de ces systèmes se trouve l'idée de répartir parmi une communauté d'usagers un ensemble important de ressources, chaque usager bénéficiant de services qui ne peuvent être obtenus que grâce à la taille de cet ensemble, à un prix rendu plus accessible par le partage.

La fonction centrale d'un système à partage de ressources est de satisfaire au mieux les demandes de ses usagers, compte tenu d'un certain nombre de contraintes. D'une façon plus précise, l'exécution d'une tâche donnée pour un usager nécessite l'utilisation d'un ensemble de ressources du système, ensemble variable avec le temps. Le rôle d'un allocateur de ressources est de fournir à chaque usager les ressources nécessaires à l'exécution de sa tâche. Cela implique une triple fonction :

- résolution des conflits entre demandes concurrentes,
- protection mutuelle des usagers,
- maintien de l'intégrité du système.

La spécification des objectifs constitue un élément fondamental du problème. Ces objectifs s'expriment en termes de services rendus aux usagers, et en termes d'utilisation des ressources. Ainsi l'objectif principal d'un système peut être de maximiser le volume total des calculs exécutés dans un temps donné ("throughput"), de maximiser le taux d'utilisation des ressources (ou de certaines ressources), ou encore de garantir à une certaine classe d'usagers la prise en compte de leur demande dans un délai maximal fixé.

Nous restreindrons notre étude à la gestion de deux classes de ressources : la mémoire et les processeurs de calcul. Par mémoire, on entend l'ensemble des supports d'information directement accessible ou non par les

processeurs.

Les problèmes d'allocation des ressources pour les opérations d'entrée-sortie ne seront pas considérés ici. Cette restriction peut être justifiée par les considérations suivantes :

- les opérations d'entrée-sortie sont en général bien découplées du reste des opérations. On cherche à réaliser un "sous-système" d'entrée-sortie prenant en charge l'ensemble des échanges pour la communauté des usagers,

- la fréquence des interactions d'un programme avec le monde extérieur est faible par rapport au cycle de base qui régit l'allocation des autres ressources. En fait, l'occurence d'une telle interaction sera essentiellement considérée dans son incidence sur le comportement du programme vis à vis des ressources "processeur" et "mémoire", (par exemple, blocage d'un processus en attente d'une entrée).

Lorsqu'on analyse les problèmes mis en jeu dans la conception d'un système à partage de ressources, on peut dégager trois domaines principaux, non entièrement indépendants :

1 - La description et l'organisation d'activités parallèles (physiquement ou conceptuellement),

2 - La gestion de l'information, c'est-à-dire son repérage par des noms et la mise en correspondance de ces noms avec les entités physiques qu'ils désignent ; la communication d'information entre le monde extérieur et le système entre également sous cette rubrique.

3 - Le partage proprement dit des ressources physiques entre leurs demandeurs concurrents, la résolution des cas de pénurie, la prévention des cas de blocage.

Tout système à partage de ressources doit comporter une solution à ces trois classes de problèmes.

1.1 Description des activités parallèles

Ainsi que nous l'avons indiqué dans l'introduction, nous nous intéressons à une classe de systèmes caractérisée par le traitement simultané (au sens macroscopique) d'un grand nombre de problèmes indépendants, et par la répartition entre des demandeurs concurrents d'un ensemble de ressources existant en quantités insuffisantes. La simultanéité, ou le parallélisme, s'observent à deux niveaux :

- physique : divers organes de traitement (processeurs, canaux d'entrée-sortie) fonctionnent en simultanéité réelle,

logique : deux programmes exécutés pour le compte de deux usagers différents se déroulent chacun de façon séquentielle, indépendamment l'un de l'autre. (de façon plus précise, pour chaque programme intervient seule la séquence des états des variables propres à ce programme, indépendamment des vitesses respectives d'exécution des deux programmes).

Pour traduire de façon commode l'existence de telles activités parallèles, les concepteurs de systèmes informatiques ont été amenés à introduire un outil : les processus. Nous commencerons par donner une idée intuitive de ce qu'est un processus avant d'en proposer une définition plus formelle.

Considérons un système d'exploitation où divers usagers peuvent faire exécuter des programmes. Pour fixer les idées, nous supposerons qu'il s'agit d'un système de traitement par trains,monoprocesseur avec multiprogrammation, permettant la simultanéité des entrées-sorties et des opérations de l'unité centrale.

Considérons dans ce système deux usagers exécutant la même tâche, que nous supposerons exécutable par un même programme réentrant, chaque usager disposant d'une zone de données propres,l'ensemble de ces informations se trouvant en mémoire. A un instant donné, les deux usagers pourront se trouver à des stades différents de leur exécution ; pour chacun d'eux, l'état instantané est défini par la donnée de la prochaine instruction à exécuter dans le programme et par la valeur des données propres à ce programme, (y compris les registres de travail). En monoprocesseur, le programme s'exécutera pour le compte d'un usager (soit 1) pendant un certain temps, l'état de l'autre (2) restant inchangé. Si 1 demande une opération d'entrée, nécessaire à la poursuite de son programme, alors l'exécution de 2 est reprise au point où elle était arrêtée ; simultanément, une opération d'entrée sera lancée par le canal pour exécuter la demande de 1 ; quand elle sera terminée, on devra trouver un moyen de le signaler, de sorte que l'exécution de 1 puisse être reprise lorsque l'unité centrale pourra lui être allouée. Plus exactement, le programme de l'usager 1 n'est pas demandeur de temps d'unité centrale tant que l'opération d'entrée n'est pas terminée, puisque ce programme ne peut progresser tant que l'information demandée n'est pas arrivée.

A partir de cet exemple très simple, nous pouvons dégager les principales notions qui nous seront utiles pour la suite.

La première remarque est que la notion seule de programme est insuffisante pour rendre compte de la situation décrite. Le programme n'est ici que la description d'un algorithme, exécutable par un processeur. Nous réserverons le nom de processus à l'entité constituée par la

succession, dans le temps, des états du système résultant de l'exécution de ce programme. L'état d'un processus à un instant donné est défini par la valeur à cet instant des diverses variables attachées au processus (dans le cas ci-dessus, ces variables ont été énumérées).

Notons que l'état ainsi défini est intrinsèque, propre au processus. Dans l'exemple ci-dessus, une description complète instantanée du système doit indiquer, outre cet état intrinsèque des processus, à quel processus est allouée l'unité centrale à l'instant considéré.

La seconde remarque est que, dans cet exemple, le déroulement d'un processus peut être arrêté pour deux raisons bien distinctes :

- l'unité centrale n'est pas disponible (elle exécute des instructions pour le compte d'un autre processus),
- le processus attend une information (plus précisément, un signal d'un autre processus lui indiquant qu'une information est disponible).

S'il y avait dans le système autant d'unités centrales que nécessaire, la première cause de blocage disparaîtrait ; toutefois, la seconde subsisterait toujours. Dans les deux cas, le blocage résulte d'une interaction entre processus ; dans le second cas, cette interaction est explicitement programmée, et fait partie intégrante de l'algorithme du processus. On parle alors d'interaction intrinsèque [S1] ou explicite [H4] ou programmée.

Dans le premier cas, l'interaction est purement contingente, elle résulte de l'insuffisance des ressources physiques. Elle n'est pas liée à l'algorithme des processus, mais à une caractéristique du système dans lequel ils s'exécutent.

On parlera ici d'interaction technologique [S1], ou implicite [H4]. Nous allons maintenant essayer de donner des définitions plus précises des divers concepts introduits.

1.2 Processus - Machines virtuelles

Un système informatique est une entité qui évolue au cours du temps entre différents états (un état étant défini par la description complète de tous les éléments physiques constituant le système, c'est-à-dire le contenu de toutes les mémoires, registres, compteurs, la situation des organes d'entrée-sortie, etc...). Le temps intervient ici non par sa mesure physique, mais comme moyen d'ordonner chronologiquement la succession des états. Pour l'objet de cette étude, on considérera les transitions du système entre deux états comme instantanées : le système est dit à évolution discrète. Le temps devient ainsi une variable discontinue.

Un système change d'état par suite de l'exécution de programmes,

c'est-à-dire d'algorithmes permettant de passer d'un état à son successeur. Nous nous attachons ici à l'exécution d'un algorithme, considérée dans son ensemble, au cours du temps. Cette exécution fait passer le système à travers une certaine succession d'états, l'algorithme comportant la définition de l'ensemble des états initiaux possibles et la définition du successeur, s'il existe, de tout état pouvant être atteint. Plus précisément, étant donné l'ensemble des états d'un système

$$S = \{E_1, E_2, \ldots\ldots\}$$

un processus P est une fonction de S dans P(S) où P(S) est l'ensemble des parties de S. (cf. [H4]).

Si P fait passer de l'état E_i à l'ensemble $F = \{E_p, E_q, \ldots\}$ et si $E_j \in F$, alors on dit que P fait passer le système de E_i à E_j, et on écrit

$$E_i \xrightarrow[P]{} E_j$$

On appelera opération toute action élémentaire d'un processus provoquant un changement d'état du système.

Si, le système se trouvant dans un état E_i, il n'existe pas d'opération faisant passer, par le processus P, dans un autre état, on dit que le processus P est bloqué en E_i.

Les définitions qui précèdent sont rigoureuses, mais encore beaucoup trop générales pour être utilisables pratiquement. Elles ont le double inconvénient de ne pas faire apparaître la distinction entre phénomènes intrinsèques et technologiques et de situer la notion d'état à un niveau de détail trop fin. Deux notions doivent être introduites : celle de ressource et celle d'interaction programmée entre processus. On verra d'ailleurs plus loin que ces deux notions ne sont pas indépendantes.

La notion de ressource peut être définie avec précision, par énumération, lorsqu'on s'intéresse à un système particulier. Pour en donner une définition générale, mais imprécise, nous dirons qu'une ressource est une entité pouvant être allouée et libérée. Une ressource allouée est associée, jusqu'à sa libération, à un processus propriétaire. Il est facile de citer des exemples de ressources : une unité centrale, une page de mémoire, etc... Une table protégée accessible à un seul processus à la fois est aussi une ressource.

L'usage d'une ressource n'est pas nécessairement restreint à un processus à la fois. On dit qu'une ressource est à N points d'accès ($N \geqslant 1$) si elle peut être simultanément allouée à N processus au plus.

Nous nous intéressons à présent, pour un processus, à l'ensemble des ressources dont il a besoin pour son exécution. Lorsqu'à un point donné de son

exécution, un processus ne peut obtenir une ressource qui lui serait nécessaire pour poursuivre, il devient alors bloqué (d'après la définition qui précède). Un tel blocage est dit technologique, ou implicite.

Un autre type de blocage est celui qui subsisterait dans l'hypothèse où il y aurait suffisamment de ressources pour satisfaire tous les besoins : c'est le blocage programmé, ou explicite, qui fait partie intégrante de l'algorithme du processus.

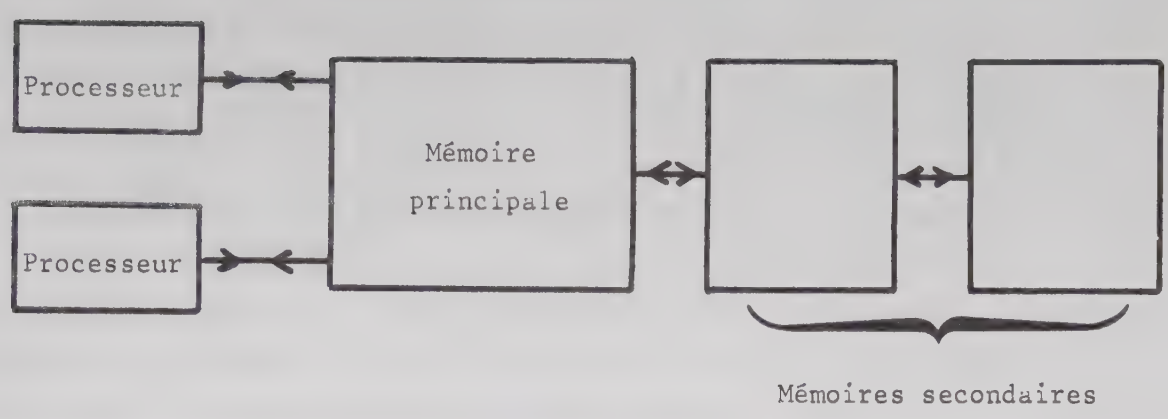
Un processus bloqué explicitement ne peut être débloqué que par l'action d'un autre processus. Nous dirons que deux processus qui n'ont pas d'interactions explicites sont indépendants.

Une fonction importante d'un système informatique est de fournir aux divers processus en activité dans ce système les ressources nécessaires à leur exécution. Cette fonction est remplie par un ensemble privilégié de processus, appelé généralement moniteur, qui utilise d'ailleurs lui-même pour sa tâche les ressources du système.

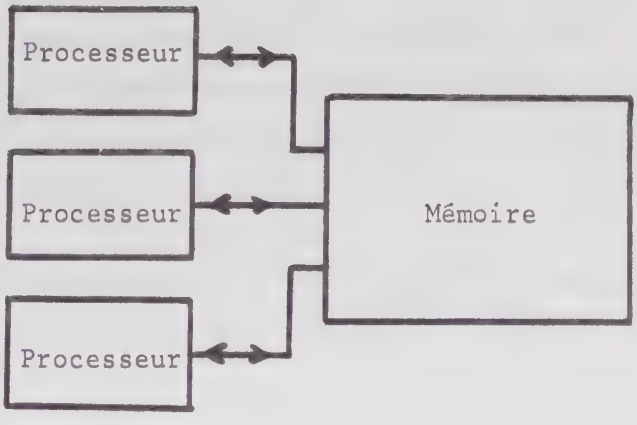
Les premiers concepteurs de systèmes à partage de ressources ont tenté de formaliser cette fonction d'allocation en introduisant la notion de machine virtuelle. On désigne sous ce terme une image du système telle qu'elle apparaît à un utilisateur à qui on fournirait un ensemble de ressources suffisant pour tous ses besoins, en lui interdisant toute possibilité d'action sur l'attribution des ressources réelles du système. La fonction d'allocation, réservée au moniteur, se traduit par une mise en correspondance dynamique (variable dans le temps) des ressources réelles du système avec les ressources virtuelles des divers utilisateurs.

Les conflits éventuels résultant du partage des ressources entre les diverses machines virtuelles sont ignorés des utilisateurs, c'est-à-dire n'apparaissent pas à l'intérieur des machines virtuelles.

On peut avoir, par exemple, la configuration suivante :



Machine physique



Machine virtuelle

Figure 1.1

Une machine virtuelle comprend au minimum une mémoire virtuelle contenant les instructions et données, et un processeur virtuel capable d'exécuter un programme, et de modifier l'état de la machine virtuelle. Nous définissons maintenant un processus comme l'exécution d'un programme sur un processeur virtuel. Il y a correspondance biunivoque entre processeur virtuel et processus.

L'état d'un processus à un instant donné peut être déterminé d'après l'état de la machine virtuelle sur laquelle il s'exécute. Plus précisément, seul un sous-ensemble de cette information est nécessaire pour caractériser l'état du processus (par exemple, dans une machine virtuelle à plusieurs processus, les données locales à un processus p n'interviennent pas dans l'état d'un autre processus q). On appelle vecteur d'état d'un processus,

l'ensemble d'informations qui détermine son état à tout instant ; le vecteur d'état comprend toujours, en particulier, les paramètres qui caractérisent l'état du processeur virtuel associé au processus (registres de travail, etc...).

La notion de vecteur d'état est importante pour la mise en oeuvre de l'allocation de ressources, qui matérialise les machines virtuelles. Lorsqu'un processus en cours d'exécution se voit retirer ses ressources, son vecteur d'état doit être conservé, pour permettre la reprise ultérieure du processus (avec, éventuellement, des ressources physiques différentes), de manière entièrement "transparente" (sans interférence avec le déroulement du processus).

Un processus peut avoir des interactions programmées avec d'autres processus associés chacun à un processeur virtuel. Le mécanisme de ces interactions n'est pas précisé pour le moment, mais doit permettre les deux opérations fondamentales suivantes :

- blocage du processus A,
- activation par le processus actif A du processus B bloqué.

On reviendra ultérieurement plus en détail sur le mécanisme d'interaction entre processus. Remarquons toutefois que l'interaction explicite entre processus n'est pas nécessairement intérieure à une même machine virtuelle. On peut en particulier avoir les deux possibilités extrêmes suivantes :

- une machine virtuelle par processus (chaque machine virtuelle possède un processeur virtuel et un seul),
- une machine virtuelle par groupe de processus indépendant (les processus d'un groupe n'ayant de relations explicites qu'entre eux).

Les avantages et inconvénients respectifs de ces deux solutions seront également discutés plus loin dans le cas particulier du système ESOPE.

L'exécution d'un processus sur une machine virtuelle se traduit par une suite de transitions de cette machine entre différents états. Du point de vue de l'allocation de ressources, nous ne nous intéressons en fait qu'à un sous-ensemble restreint de ces transitions : celles qui correspondent au blocage ou au déblocage d'un processus. Seul un processus actif, en effet, est un demandeur de ressources. Pour nous, désormais, l'activité intrinsèque d'un processus, c'est-à-dire son exécution sur un processeur virtuel, se traduit par une suite de transitions entre les états actif et bloqué (états "intrinsèques")

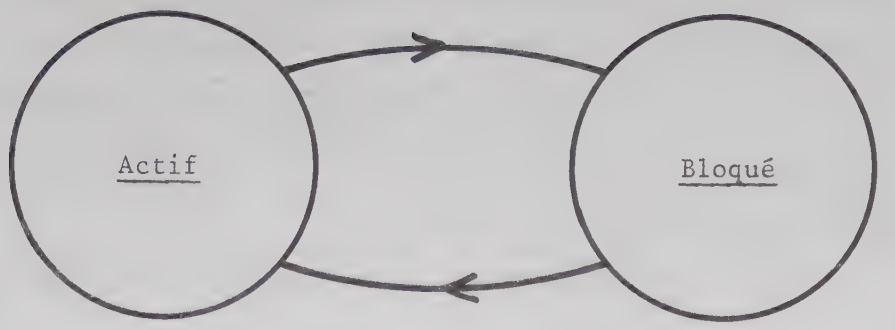


Fig. 1.2 : Etats intrinsèques d'un processus

Si nous considérons maintenant la situation globale du système en faisant intervenir l'allocation des ressources, nous voyons que le déblocage d'un processus ne se traduit pas nécessairement par son exécution immédiate sur un processus réel : cela dépend de l'état d'allocation des ressources à la machine virtuelle associée à ce processus. Il est alors commode de définir pour le processus un ensemble d'états (dits "effectifs") traduisant sa situation réelle compte tenu de la répartition des ressources. Deux types d'événements peuvent déclencher une transition entre états :

- événements intrinsèques : blocage ou activation du processus,
- événements technologiques : modification du nombre ou de la nature des ressources allouées au processus (par l'intermédiaire de sa machine virtuelle).

Chaque état d'un processus est défini par :

- la situation intrinsèque du processus (actif ou bloqué)
- les ressources allouées au processus (éventuellement)
- les ressources demandées par le processus

Une politique d'allocation de ressources est définie par l'ensemble des algorithmes permettant de réaliser les transitions entre les états effectifs des processus ; cette définition comporte en particulier, la spécification de la structure des ensembles de processus se trouvant dans le même état effectif. De tels ensembles sont souvent gérés comme des files d'attente (processus attendant une même ressource, par exemple), mais ce n'est pas obligatoire. Par ailleurs, on doit spécifier les règles n'appliquer en cas de variation du nombre de processus dans le système, par création ou destruction.

Chaque état d'un processus comporte la spécification des ressources allouées à ce processus. On voit qu'on peut définir, corrélativement aux états des processus, des états pour les ressources ; les transitions entre états de

processus correspondent à des transitions entre états de ressources.

La description d'un système d'exploitation sous forme d'un double diagramme d'états des processus et des ressources, accompagné des règles de transition, fournit un outil commode pour la conception et la réalisation des systèmes. Cette méthode sera appliquée dans la seconde partie pour la description complète du système ESOPE.

Chapitre 2

MODELES DE COMPORTEMENT DES PROGRAMMES ET DES USAGERS

On s'intéresse dans ce chapitre au comportement des usagers d'un système informatique, considérés comme des générateurs de demandes de processeur et de mémoire. Le but recherché ici est de dégager des principes suffisamment généraux pour guider l'élaboration d'une stratégie d'allocation de ces ressources.

L'exécution d'un programme est considérée, du point de vue de la demande de ressources, comme une succession de références à des adresses virtuelles. Cette succession caractérise intrinsèquement le programme et c'est elle qui détermine ses besoins en mémoire physique. La transformation des références aux adresses virtuelles en demandes de mémoire physique dépend toutefois de la manière dont est établie la correspondance virtuel-physique (mécanisme) et également de la manière dont est gérée la ressource mémoire (stratégie).

Les champs d'investigation sont donc les suivants :

- étude et modèles du comportement intrinsèque d'un programme
- étude et modèles du comportement d'un programme
    - en fonction du mécanisme de correspondance virtuel-physique
    - en fonction de la stratégie d'allocation de la mémoire.

2.1 Comportement intrinsèque des programmes

L'étude du comportement intrinsèque d'un programme présente un double intérêt :

- permettre de construire des modèles de simulation pour l'étude de données stratégies,
- fournir une base théorique à l'étude du comportement de programmes dans un environnement donné.

L'exécution d'un programme se présente comme une suite de références à des adresses contenues dans un domaine donné. Ces références résultent de l'exécution d'instructions de programmes ou d'accès à des données.

La propriété fondamentale observée dans le comportement intrinsèque des programmes est d'ordre qualitatif ; elle a été observée expérimentalement de longue date, et se justifie par un ensemble de considérations liées pour une grande part à la structure séquentielle des machines du type de Von Neumann. On peut caractériser le comportement des programmes par la tendance à une certaine "rémanence" dans la succession des accès à l'information. Autrement dit, l'observation du comportement d'un programme pendant un certain temps fournit en moyenne une quantité d'information non nulle sur son comportement dans un futur peu éloigné. La probabilité qu'un programme accède à une information "voisine" d'une information référencée dans un passé proche est plus grande que la probabilité a priori (c'est-à-dire dans l'ignorance du comportement passé) de cet événement. La notion de voisinage est définie en termes de différences d'adresse, dans l'hypothèse d'une machine à adressage séquentiel.

On peut énumérer les principales raisons de ce comportement :

- la nature séquentielle (déjà mentionnée) des programmes, dans les machines actuelles : l'instruction exécutée après l'instruction d'adresse x est l'instruction d'adresse x + 1, sauf si l'instruction x agit sur le compteur ordinal. Or on observe que la proportion des instructions de saut dans le programme est de l'ordre de 1/3, et souvent moins dans les programmes produits par des compilateurs. Parmi ces sauts, une proportion importante sont en fait des instructions contrôlant une boucle ; et les boucles sont en moyenne de faible dimension (par rapport à la taille totale des programmes), ce qui contribue à la tendance au comportement stationnaire.

- La nature séquentielle de l'accès aux données. Sauf dans des cas tels que les langages de liste, où une grande partie des données manipulées sont des pointeurs, c'est-à-dire des voies d'accès à d'autres informations, l'accès aux données à tendance à être séquentiel (scrutation de tables, lecture de fichiers). Par ailleurs, les langages à structure de bloc imposent par leur nature même la stabilité des accès aux données, puisque le domaine de données accessible n'évolue que lors des entrées et sorties de blocs ou de procédure.

- Enfin, le phénomène est amplifié par le fait que les systèmes d'exploitation favorisent (parfois très fortement) les programmes à comportement stable (références peu dispersées et domaine d'accès variant lentement). Les programmeurs et les constructeurs de compilateurs sont ainsi incités à

produire des programmes ayant ces caractéristiques.

L'étude expérimentale du comportement des programmes a surtout été faite dans le cadre d'un mécanisme d'adressage donné (segmentation, pagination) et de contraintes imposées (stratégies d'allocation). Nous reviendrons plus loin en détail sur les résultats de ces expériences. Des modèles théoriques ont été proposés par divers auteurs (cf[P1]). Dans tous ces modèles, un programme est considéré comme un graphe orienté, dont les noeuds constituent des séquences d'instructions ou de données. A chacun des arcs est attachée une probabilité de transition. L'utilité de tels modèles réside dans la détermination du découpage optimal d'un programme en segments de manière à réduire les transitions entre segments, moyennant certaines contraintes (taille des segments). Toutefois, la complexité des algorithmes d'optimisation semble restreindre de telles applications au cas de programmes très fréquemment utilisés, et exclure leur utilisation en temps réel dans un système d'allocation de mémoire.

2.2 Comportement des programmes en mémoire paginée : résultats expérimentaux

Nous réservons une place particulière à l'étude du comportement dynamique des programmes en mémoire paginée pour les raisons suivantes :

- la plupart des résultats expérimentaux disponibles concernent ce type de mémoire,
- le système ESOPE est construit pour une machine disposant d'une mémoire paginée.

2.21 Expériences de SDC

Les premières expériences systématiques ont été menées en 1966 par une équipe de SDC (Systems Development Corporation) sur le système de temps partagé de la machine AN/FSQ-32. [F2]

L'expérience montre que les programmes suivent sensiblement un même comportement :

a - des séquences courtes d'instructions exécutées sur chaque page.

b - Un nombre considérable de références à des pages de données, par séquence d'instructions.

c - Des références aux pages de données très dispersées dans le temps.

Les résultats présentés ont été obtenus dans les conditions suivantes : un programme croît librement jusqu'à la fin du quantum (400 ms) ou

jusqu'au moment où il appelle le moniteur ; ce laps de temps est désigné, par les auteurs, sous le nom d'intervalle de service ; les pages sont chargées à la demande, et pendant un intervalle de service toute page référencée est disponible pour le restant de l'intervalle.

Donnons une idée de la brièveté de l'intervalle de service : dans 60 % des cas, moins de 20 instructions sont exécutées, dans 80 %, moins de 200.

La figure ci-dessous, extraite de [F2], présente le nombre moyen de pages référencées en fonction du temps, mesuré depuis le début d'un intervalle de service.

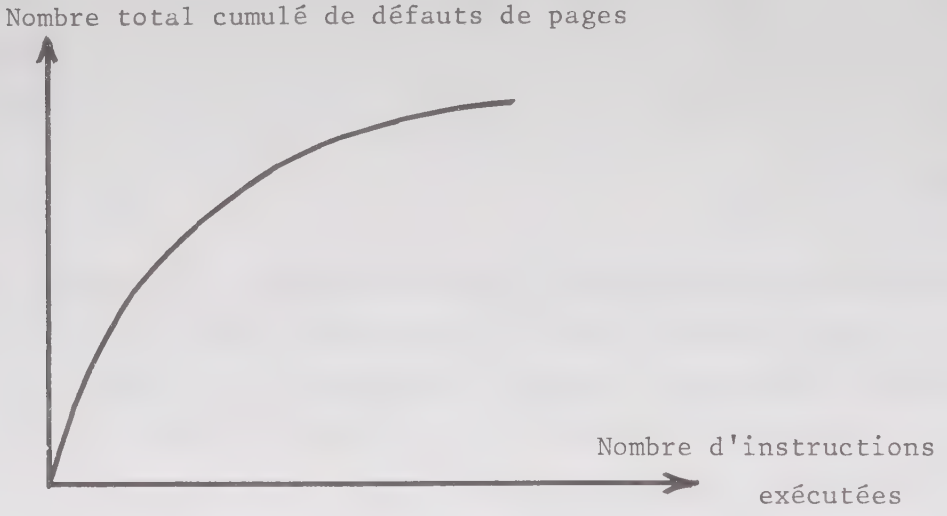


Figure 2.1

2.22 Expériences de Princeton

A partir de résultats de mesures accumulés pendant l'exploitation de plusieurs programmes représentant un ensemble assez large de types de comportement (prédominance de calcul, données dispersées ou non...). Coffman et Varidn [C3] ont entrepris une simulation pour mesurer les caractéristiques du trafic de pages suivant différents algorithmes de remplacement, et aussi en utilisant plusieurs tailles de pages. Les pages sont chargées à la demande ; un programme peut disposer de N pages en mémoire, paramètre de la simulation, et lorsqu'elles sont toutes occupées, on fait appel à un algorithme de remplacement.

Les principaux résultats de cette étude peuvent être résumés ainsi :

- la variation du nombre de défauts de pages avec la taille de mémoire est non-linéaire (il y a croissance très rapide au-dessous d'un certain seuil). Ce résultat est en accord avec ceux de [F2].

- à taille de mémoire donnée, il est plus rentable de multiplier par 2 le nombre de pages plutôt que de doubler la taille des pages (pour des pages de

1 K mots). Par ailleurs, toujours pour des pages de 1 K mots, le nombre moyen d'instructions exécutées entre deux défauts de pages varie entre 300 (lorsque la mémoire contient 8 pages) à 60 (lorsqu'elle n'en contient que 2). On peut ainsi avoir une idée quantitative du caractère "local" du comportement des programmes.

2.23 Expériences d'IBM en mémoire restreinte

Les expériences de Brawn et Gustavson [B6] entrent dans le cadre d'une étude systématique sur le comportement des programmes et les stratégies de remplacement en mémoire paginée, étude menée sur un système expérimental, le M44/44X. Ce système simule plusieurs machines virtuelles, la mémoire étant allouée par pages.

Les courbes de la figure 2.2 extraites de [B6] indiquent en fonction de la taille de mémoire réelle, le nombre de défauts de pages enregistrés au cours de l'exécution d'un programme (inversion de matrice) exécuté seul (en monoprogrammation). Deux versions du même programme ont été exécutées ; l'une programmée sans tenir compte du fait que le programme s'exécutait dans une mémoire paginée, l'autre tenant compte de ce fait et essayant de réduire le nombre de défauts de pages, en évitant les références dispersées dans le programme et les données.

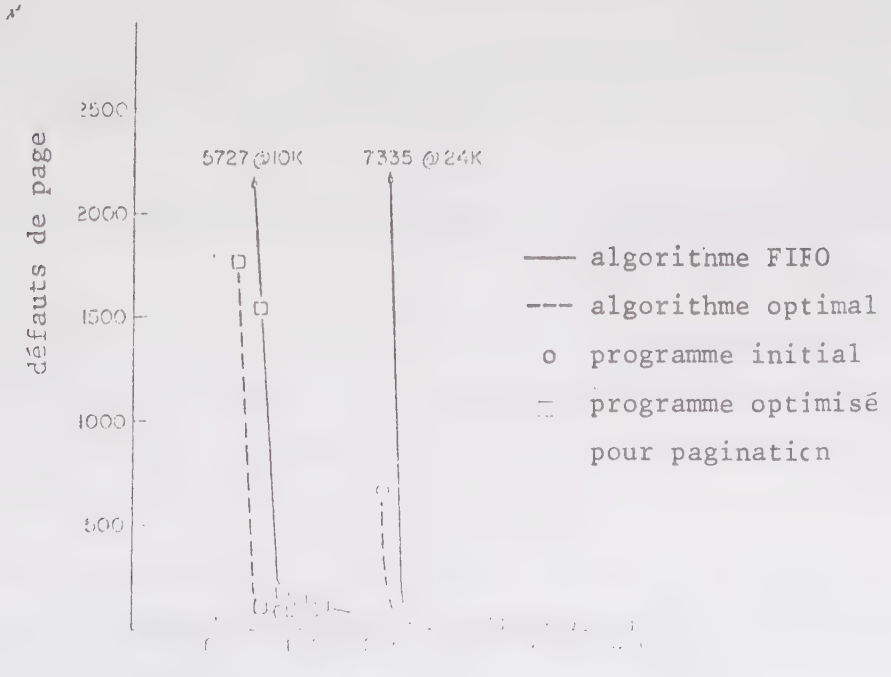


Figure 2.2

Par ailleurs, chacune des deux versions du programme a été exécutée en utilisant successivement deux algorithmes de remplacement de pages : un algorithme minimisant le nombre de pages chargées (cet algorithme, du à Belady, n'est pas réalisable car il suppose connue entièrement la séquence des références aux pages ; il nécessite donc une passe préalable, et sert ici d'étalon de comparaison), et l'algorithme FIFO (remplacer la page la plus anciennement chargée), simple à réaliser mais n'ayant aucune vertu particulière.

Les courbes de la figure 2.2 (et les résultats analogues donnés dans [B6]) donnent lieu à trois remarques :

1) la non-linéarité de la variation du nombre de défauts de page avec la taille de la mémoire est très notable. Lorsqu'on réduit la taille de la mémoire, la dégradation des performances est peu sensible jusqu'à une certaine taille, mais explosive au-dessous de cette taille (que l'on appelle parfois "parachore" du programme). La notion de parachore est qualitative, elle n'est applicable qu'aux programmes présentant un comportement du type indiqué sur la fig. 2.2. Suivant le programme, la chute de performances est plus ou moins rapide.

2) Il est possible d'améliorer considérablement le comportement d'un programme si on tient compte du fait qu'il doit s'exécuter dans une mémoire paginée. Cette remarque est surtout importante pour les programmes communs fréquemment exécutés.

3) Vis à vis des deux effets ci-dessus, l'influence de l'algorithme de remplacement des pages est pratiquement négligeable

2.3 Comportement des programmes en mémoire paginée : modèles théoriques

2.31 Modèle du "working set"

Le modèle du "working set" a été introduit par DENNING ([D2]) pour rendre compte de la propriété expérimentalement observée de comportement local des programmes et utiliser cette propriété pour définir une stratégie de remplacement.

S'agissant d'un modèle de comportement intrinsèque, on s'intéresse au fonctionnement d'une machine virtuelle constituée d'un processeur et d'une mémoire paginée. Le "working set" d'un processus à l'instant t est défini comme l'ensemble $W(t,\tau)$ des pages virtuelles auxquelles le processus a accédé au cours de l'intervalle de temps $[t-\tau, t]$. La taille du working set exprimée en nombre de pages est désignée par $\Omega(t,\tau)$. On montre aisément [D2] que c'est une

fonction croissante et convexe de τ.

La propriété de comportement local s'exprime par le fait que les références d'un programme à ses pages dépendant des références faites dans un passé peu éloigné, et donc que $W(t,\tau)$ est une bonne estimation des références du processus dans un futur "proche" (c'est-à-dire de l'ordre de grandeur de τ).

On peut donner [D4] une expression plus précise de la propriété de comportement local qui donne son intérêt à la notion de "working set". Soit :

$$r_1, r_2, \ldots r_k, \ldots \qquad 1 \leqslant r_i \leqslant N$$

La séquence de références d'un processus à ses pages virtuelles. Soit :

$$a_i(k) = Pr(r_k = i) \qquad 1 \leqslant i \leqslant N$$

la probabilité pour que la k ième référence à une page distincte concerne la page i. On a :

$$0 \leqslant a_i(k) \leqslant 1$$

$$\text{et} \quad \sum_{i=1}^{N} a_i(k) = 1$$

Soit alors $R(k)$ la permutation $(i_1, i_2, \ldots i_N)$ telle que l'on ait

$$a_{i_1}(k) \geq a_{i_2}(k) \geq \ldots \geq a_{i_N}(k)$$

La propriété de comportement local s'exprime alors par les deux conditions qualitatives suivantes :

1) l'ordre entre les $a_i(k)$ est strict.

2) la durée de vie moyenne d'une permutation $R(k)$ (nombre de références entre deux modifications de la permutation) est grande. Autrement dit, les $a_i(k)$ varient lentement avec k.

Le "working set" $W(k,h)$ à la k-ième référence est maintenant défini comme l'ensemble des numéros des h dernières pages référencées antérieurement. La valeur du "working set" en tant que prédicteur du comportement futur peut s'exprimer qualitativement en remarquant que les pages ayant un $a_i(k)$ élevé (donc ayant une grande probabilité d'accès à la k-ième référence), ont également,

d'après la stabilité de R(k), une probabilité élevée d'avoir été référencées récemment et donc de figurer dans le "working set". La probabilité pour que la k-ième référence concerne une page se trouvant dans le "working set" W(k, h) est donnée par :

$$q(k,h) = \sum_{i \in W(k,h)} a_i(k)$$

On peut alors exprimer de façon quantitative la propriété de comportement local d'un programme, par la fonction h(k, p) définie comme la plus petite valeur de h satisfaisant à la condition :

$$q(k, h) \geqslant p \qquad\qquad p \in [0,1]$$

A k donné, h est une fonction non décroissante de p, nulle pour p = 0, et atteignant un maximum égal à k pour p = $p_o(k) = \sum_{i \in W(k,k)} a_i(k)$

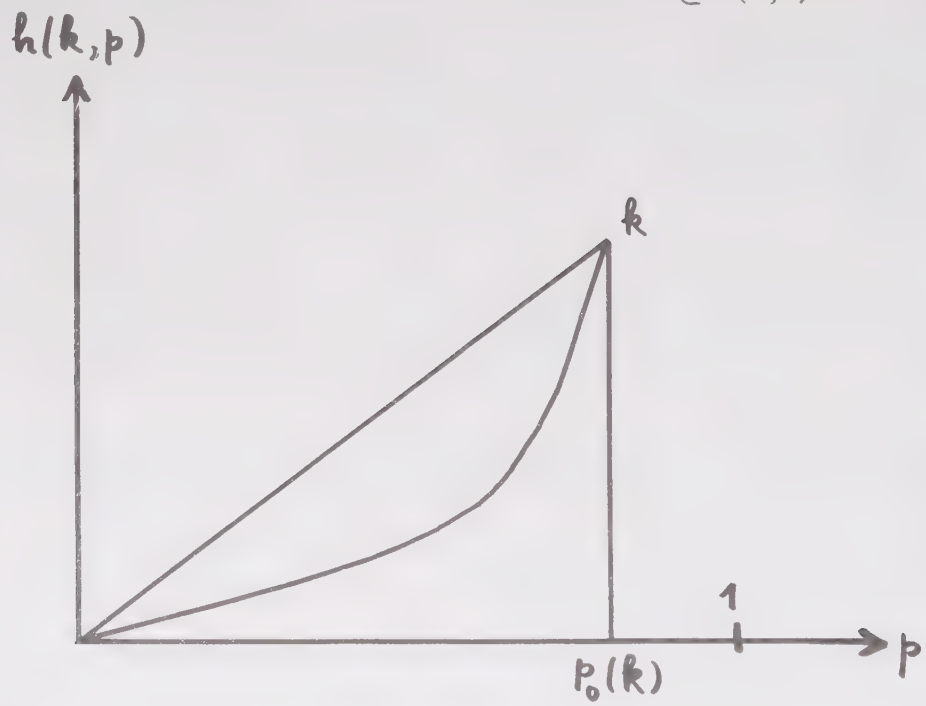


Fig. 2.3

h croît linéairement avec p pour un programme dont les références sont uniformes ($a_i = 1/N$, $p_o(k) = k/N$). La tendance au comportement local est d'autant plus grande que la croissance initiale de k avec p est plus lente.

On peut expliquer qualitativement au moyen du schéma ci-dessus le

comportement d'un programme en mémoire restreinte. Soit en effet un programme s'exécutant dans une mémoire de M pages. Si $M \geqslant h(k, q_o)$, (pour un qo donné), alors la probabilité P de défaut de page à la k-ième référence est donnée par $q(k, h)$ et elle est inférieure par définition à $1-q_o$. Si $M < h(k,q_o)$, alors la dépendance de p vis à vis de M se manifeste. En effet, on a

$$p = 1 - q_o + q_o \frac{h(k,q_o) - M}{h(k,q_o)} = 1 - q_o \frac{M}{h(k,q_o)}$$

(en admettant que l'algorithme de remplacement choisisse au hasard la page à remplacer).

On voit que si la tendance au comportement local est forte, alors on pourra avoir un h notablement inférieur à N pour q_o élevé, et le programme pourra continuer à s'exécuter avec peu de défauts de pages dans une mémoire de taille notablement inférieure à la sienne propre. On peut interpréter de cette manière des résultats tels que ceux donnés au § 3.23.

2.32 Modèles statistiques

L'étude de la séquence de références aux pages d'un programme $\{r_i\}$ peut être faite de deux façons :

- par observation du déroulement de programmes réels, en utilisant un programme de "trace". Cette méthode est utilisable pour la collecte des données d'une étude de simulation.
- en établissant un modèle statistique du comportement de la suite $\{r_i\}$.

Cette méthode est utilisable pour la simulation, et se prête également aux études analytiques.

Nous nous intéressons ici à la seconde méthode. Nous montrons comment un modèle simple peut être utilisé pour rendre compte des phénomènes observés expérimentalement.

2.321 Modèle à probabilités constantes

Etant donné les N pages virtuelles d'un programme, numérotées de 1 à N, le modèle de comportement, le plus simple consiste à attacher à chacune de ces pages une probabilité d'occurence p_i, avec

$$\sum_{i=1}^{N} p_i = 1$$

p_i mesure la probabilité pour la page n° i de faire l'objet de la k-ième référence. Cette probabilité est constante dans le temps, donc indépendante de k.

$$Pr\ (r_k = i) = p_i$$

Un tel modèle peut être utilisé pour représenter de façon approximative le phénomène de dégradation de performances en mémoire restreinte ; pour cela, il suffit de choisir un sous-ensemble de pages (par exemple les pages 1 à P) à chacune desquelles on attribue une probabilité p, les autres pages recevant la probabilité q

$$Pp + (N - P)q = 1 \qquad p \geqslant \frac{1}{2}$$

En faisant exécuter le programme en mémoire restreinte, on obtient la courbe de réponse représentée sur la figure 2.4, qui représente assez correctement les résultats du type de ceux de [B6] ou de [M2].

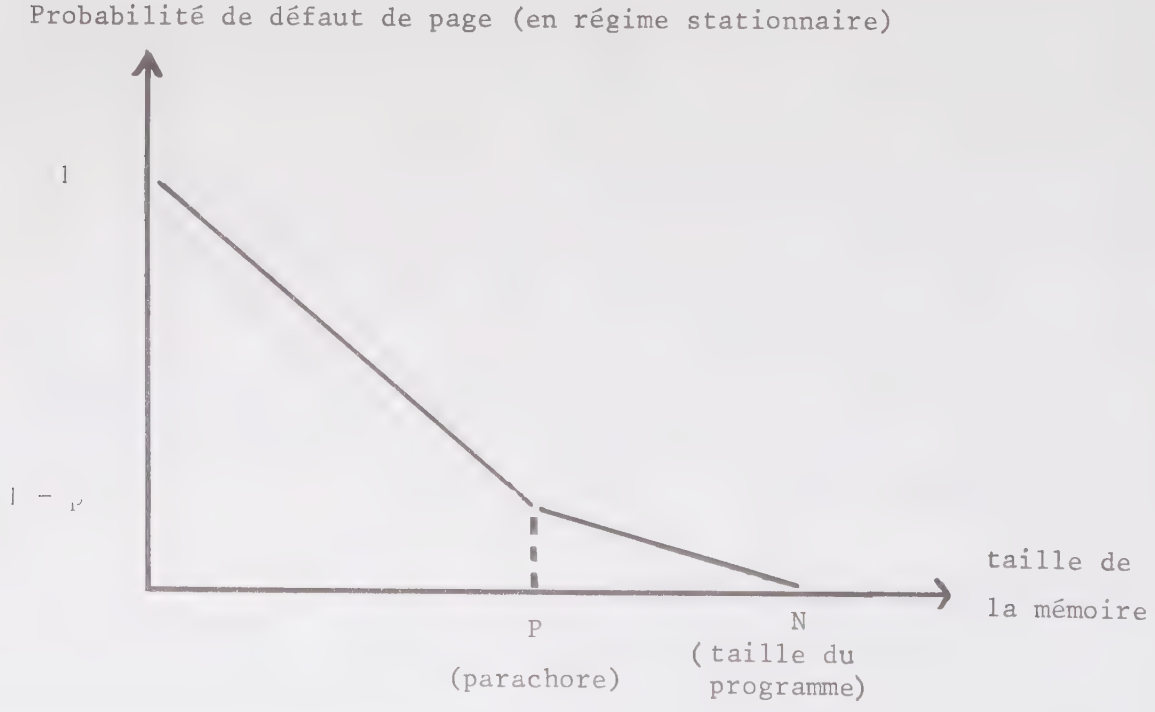


Fig. 2.4

Pour introduire une terminologie qui sera utilisée dans la 3ème partie, nous dirons que les P pages de probabilité d'occurence $p > 0.5$ représentent un "noyau" du programme dont le paramètre $pP/(1-Pp)$ mesure la "dureté".

Avec ce modèle, un programme est caractérisé par deux nombres sans dimension : P/N, rapport du noyau à la taille totale du programme, et la dureté $pP/(1-pP)$. Plus un programme est "dur", et plus ses références sont localisées ; à l'inverse un programme dont la dureté est voisine de 1 dispersera uniformément ses références.

Un modèle à probabilités constantes a été utilisé dans [K4] pour étudier un schéma d'algorithmes de remplacement. Les lois choisies étaient du type uniforme, arithmétique ou géométrique ; le résultat le plus significatif de cette étude est que le nombre de défauts de page est beaucoup plus sensible à la taille de mémoire qu'à l'algorithme de remplacement utilisé.

2.322 Modèle markovien stationnaire

On considère cette fois comme constantes non les probabilités d'occurence mais les probabilités de transition p_{ij} entre pages, c'est-à-dire

$$p_{ij} = Pr\ (r_{k+1}=j \mid r_k=i) \qquad i,j \in [1,N]$$

avec :
$$\sum_{j=1}^{N} p_{ij} = 1 \qquad i \in [1,N]$$

La matrice de transition p_{ij} sera notée P.

Nous allons établir quelques propriétés de ce modèle. Les résultats de la théorie des chaînes de Markov que nous utiliserons sont énoncés et établis dans [K1] . Nous supposerons que toute page du programme fait l'objet d'une référence au bout d'un temps suffisant, et qu'il n'y a pas d'"état d'absorption" (c'est-à-dire que $p_{ii} \neq 1$). Ces deux conditions se traduisent par l'hypothèse que la chaîne de Markov représentant le comportement du programme est régulière.

La probabilité d'occurence s'exprime par

$$P_r(r_{k+1} = i) = \sum_{j=1}^{n} p_{ij}^{(k)} \; b_j$$

où $\{b_i\}$ est le vecteur des probabilités initiales (b_i = probabilité pour que la première référence soit à la page i), et $p_{ij}^{(k)}$ représente l'élément (i,j) de la matrice P^k.

On voit donc que le comportement des puissances successives de la matrice de transition joue un rôle déterminant dans ce modèle. Ce comportement est régi par le résultat fondamental suivant [K1] :

Pour toute chaîne de Markov régulière, les puissances successives P^n de la matrice de transition convergent vers une matrice limite A, dont les lignes sont identiques à un même vecteur a. La convergence est géométrique. Le vecteur a est la solution unique (à un facteur constant près) de l'équation

$$aP = a$$

Le vecteur a est un vecteur de probabilité ($\sum a_i = 1$), qui conditionne le comportement asymptotique de la chaîne de Markov, en ce sens que

$$\lim_{k \to \infty} Pr(r_k = i) = a_i$$

la convergence étant géométrique, donc très rapide.

L'utilisation d'un modèle markovien au lieu d'un modèle à probabilités constantes paraît donc surtout justifiée si on s'intéresse de manière assez fine au comportement des programmes ou si on se limite à un nombre assez faible de transitions. Ce second cas se présente si on s'intéresse aux transitions non entre pages mais entre blocs ("modules").

2.323 Modèles à caractéristiques variables

Les modèles examinés jusqu'ici avaient un comportement déterminés par des paramètres invariants au cours du temps. En fait, sur une période de temps assez longue, on constate que les caractéristiques des programmes réels varient. En particulier, la taille du "working set" ne reste pas constante ; des études récentes ([C3] ,[D4]) introduisent un modèle où la taille du "working set"

est une variable aléatoire normale stationnaire. Ce modèle peut être justifié par des considérations théoriques. Il est aisé d'étendre le modèle à "noyau" défini au § 2.321 en considérant la taille P du noyau comme une variable aléatoire normale. Ce modèle sera utilisé pour les expériences de simulation décrites dans la 3ème partie.

2.4 Comportement des usagers conversationnels

Dans un système où la charge principale provient de travaux en mode conversationnel, la demande de processeur est gouvernée par le comportement des utilisateurs dans ce mode de travail. Ce comportement a fait l'objet de mesures et d'études statistiques, en nombre d'ailleurs assez réduit. On présente ici une synthèse de ces résultats.

2.41 Généralités

Un usager, devant sa console, n'est pas toujours occupé à frapper des messages ; le cycle de son activité se déroule ainsi : il réfléchit à sa demande, frappe son message, qui est soit une commande au moniteur ou à un traducteur, soit une donnée pour un traducteur ; il attend alors la réponse, examine le résultat, et le cycle recommence.

Nous appelerons interaction l'intervalle de temps défini par deux messages consécutifs. Il est évident que tous les intervalles de temps que nous allons considérer sont des grandeurs aléatoires, grandeurs qui fluctuent non seulement d'un individu à un autre, mais encore, pour un individu donné, évoluent avec la nature du programme considéré. En revanche, l'expérience montre que le comportement moyen d'un usager est indépendant du nombre d'utilisateurs en conversation, de l'heure de travail (travail nocturne) et même du jour (travail de week-end).

Une interaction peut être décomposée dans le temps de plusieurs façons ; Il est possible (comme dans[S2]) de lier cette décomposition à la structure d'un système donné, mais cela limite l'applicabilité des résultats. Une décomposition simple et non ambiguë consiste à distinguer la partie dépendant uniquement du système et celle correspondant à l'activité de l'usager (fig. 2.5).

La première partie, appelée temps de réponse comprend l'exécution proprement dite du message, après une attente (en général dans une file), et la sortie éventuelle d'une réponse ; il n'est pas toujours possible de séparer dans le temps l'exécution et la sortie du résultat, car ces deux opérations peuvent se réaliser en parallèle. Ce temps de réponse est une caractéristique

du système et un des paramètres les plus utiles pour évaluer ses performances.

La partie dépendant de l'usager, ou temps de préparation, débute une fois la réponse complètement sortie. L'utilisateur examine le résultat, réfléchit et frappe le message suivant ; réflexion et écriture se déroulent en parallèle et il semble artificiel de distinguer, comme on le fait souvent, un temps d'écriture et un temps de réflexion. La durée de ce temps de préparation est uniquement fonction de l'usager, car le débit maximal de la console (de l'ordre de 10 à 15 caractères par seconde) ne ralentit pas l'entrée du message.

D'autres caractéristiques importantes des interactions sont :

la longueur des messages, la répartition des messages en commandes et données, la fréquence d'utilisation des commandes, la taille du programme exécuté.

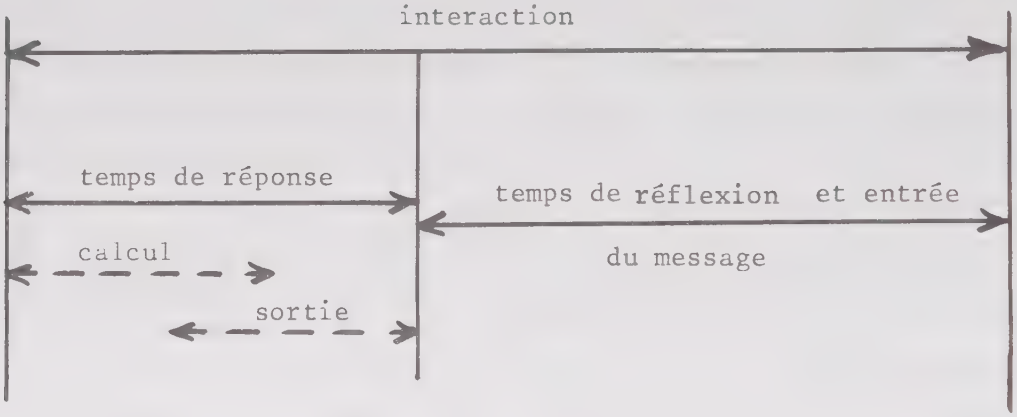


Fig. 2.5 : Comportement d'un usager conversationnel

Paramètres caractéristiques.

Des principaux résultats publiés, on peut dégager les conclusions suivantes :

- la durée moyenne du temps de réflexion s'établir autour d'une trentaine de secondes (avec un important écart type d'une vingtaine de secondes), pour des systèmes aussi différents que JOSS et CTSS .
Cette dispersion s'explique par la variété des tâches correspondant à une interaction (assemblage, édition, manipulation de fichiers).

- la taille des programmes mis en jeu par une interaction et leur durée sont très fortement dispersées. Une distribution exponentielle peut rendre compte d'un type donné d'interaction.

- un terminal conversationnel est oisif environ la moitié du temps.

2.5 Conclusions pratiques

Essayons de tirer les conséquences pratiques de ce qui précède, en ce qui concerne l'allocation des ressources dans un système :

- le comportement d'un programme dans sa mémoire virtuelle est caractérisé par la non-uniformité des références, et en particulier une tendance à un comportement "local". Plusieurs modèles de comportement permettent de rendre compte de cette tendance ; ces modèles donnent des renseignements statistiques et sont peu utilisables pour la prévision précise du comportement d'un programme donné,

- lorsque la mémoire virtuelle se trouve matérialisée par une mémoire à accès direct et une ou plusieurs mémoires secondaires, le facteur déterminant dans le nombre de transferts d'information entre niveaux de mémoire est constitué par les caractéristiques du comportement des programmes plutôt que par la stratégie de gestion de la mémoire,

- en particulier, le comportement d'un programme dans une mémoire paginée restreinte est caractérisé par une variation non-linéaire du nombre de défauts de page avec la taille de la mémoire ; ce nombre croît rapidement quand cette taille tombe au-dessous d'un seuil dépendant du programme (parachore). Corrélativement, le taux d'utilisation du processeur par ce programme décroît, et d'autant plus rapidement que le temps de transfert entre mémoire principale et mémoire secondaire est élevé.

Il semble donc utile de pouvoir déterminer ce seuil d'efficacité pour un programme. On peut utiliser pour cela la propriété d'extrapolation du modèle du "working set", et prendre les pages référencées dans un passé récent comme estimation des pages utilisées dans un proche avenir. On doit pour cela définir le paramètre τ du "working set", c'est-à-dire la durée d'observation ; ce paramètre dépend des caractéristiques du programme (stabilité du "working set"), de l'efficacité recherchée, et du temps de transfert entre niveaux de mémoire. En effet, dans l'hypothèse d'un comportement stationnaire, si $m(\tau)$ est la probabilité de défaut de page à l'instant t lorsque les seules pages de $W(t,\tau)$ sont en mémoire, l'efficacité (taux d'utilisation du processeur) pour un programme est

$$e(\tau) = \frac{1}{1 + x.m(\tau)}$$

où x est le rapport du temps de transfert mémoire centrale-mémoire secondaire à la durée moyenne d'un accès à la mémoire principale. Pour obtenir au moins une efficacité égale à e_o, on devra donc choisir τ, durée d'observation du "working

set", tel que $m(\tau) < \frac{1-e_o}{e_o x}$. $m(\tau)$ est une fonction décroîssante de τ (d'autant plus vite que les références du programme sont plus groupées). On a ainsi une idée de la forme de la relation entre "working set" et parachore.

Dans le chapitre suivant, on examinera comment on peut tenir compte des caractéristiques du comportement des programmes dans la mise en oeuvre des stratégies d'allocation des ressources.

Chapitre 3

ALGORITHMES DYNAMIQUES D'ALLOCATION

Le but de ce chapitre est de présenter l'"état de l'art" en ce qui concerne les stratégies de régulation dynamique de l'allocation des ressources dans les systèmes multiprogrammés. Une telle stratégie est utilisée dans le système ESOPE et sera présentée dans la seconde partie (ch. 6) ; des résultats expérimentaux seront donnés dans la troisième partie (ch. 8).

Les méthodes de régulation dont il sera ici question ont toutes pour but d'éviter un phénomène constaté lors du développement des premiers systèmes multiprogrammés, notamment ceux faisant appel à l'allocation de mémoire par "page à la demande". [F2]. Une analyse en fut donnée en 1968 [D3] sous le nom de "thrashing" (ou "écroulement") analyse dont nous rappelons ici le principe. Toutefois peu nombreux sont encore les systèmes d'exploitation comportant une stratégie d'allocation de ressources orientée vers la prévention de ce phénomène, et plus rares encore les résultats expérimentaux relevés lors du fonctionnement de tels systèmes.

3.1 Le phénomène d'écroulement ("thrashing")

On a donné le nom d'écroulement d'un système d'exploitation à un phénomène caractérisé par les symptomes suivants :

- forte activité (ou même saturation) du canal de transfert entre mémoire principale et mémoire secondaire,
- faible activité de l'unité centrale,
- accroissement considérable du temps de réponse moyen.

La politique d'allocation de pages "à la demande" a initialement été incriminée. Ultérieurement, l'écroulement d'un système a été reconnu comme une conséquence d'une politique trop libérale d'allocation de mémoire, et expliqué, au moins qualitativement, en fonction du comportement des programmes en mémoire restreinte. Nous rappelons brièvement ici le principe de cette analyse. [D3]

On étudie l'évolution d'un système comportant une mémoire centrale, une mémoire secondaire et un processeur. Ce système sert n travaux (ou processus) que l'on suppose avoir des caractéristiques identiques. On considère l'un de ces travaux à qui est alloué pendant une certaine période, une taille moyenne m de mémoire ; soit f(m) la probabilité moyenne de défaut de page pour ce programme pendant cette période.

Le nombre moyen d'accès à la mémoire entre deux défauts de page est 1/f(m). Si on appelle e(m) la fraction du temps virtuel passée par le processus à s'exécuter, on a :

$$e(m) = \frac{\text{temps d'exécution}}{\text{temps d'exécution + temps d'attente de page}}$$

soit

$$e(m) = \frac{t\ (1/f(m))}{t/f(m) + T} = \frac{1}{1 + x\, f(m)}$$

où t est la durée moyenne d'un accès à la mémoire principale

T est le temps moyen de transfert d'une page

$x = T/t$

La valeur numérique de x est de l'ordre de 10^3 à 10^4 pour les mémoires usuelles. On voit donc que si m est tel que f(m) varie rapidement avec m (c'est à dire au voisinage du parachore), une petite variation de m provoquera une variation considérable de e(m).

Supposons que la répartition de la mémoire totale entre les n demandeurs soit telle que chacun dispose d'un nombre de page voisin de son parachore, et qu'un (n + 1) ème programme soit introduit. Si la politique d'allocation de mémoire ne comporte aucune restriction particulière, le nombre de pages alloué à chacun des programmes présent sera uniformément réduit ; d'après la forme de la fonction e(m), tous les programmes verront leur efficacité tomber rapidement : c'est le phénomène d'écroulement.

On voit que les facteurs déterminants de l'écroulement sont :

- les caractéristiques moyennes du comportement des programmes (existence du parachore),

- la forte disparité des temps d'accès à la mémoire principale et à la mémoire secondaire,

- une politique "globale" d'allocation de mémoire tendant à allouer des pages sans restriction, au détriment éventuel de programmes déjà en mémoire.

C'est sur ce dernier facteur que l'on devra agir pour éviter l'écroulement.

3.2 Prévention de l'écroulement

L'analyse qui précède a permis de préciser les causes de l'écroulement. Elle en suggère également les remèdes. Il s'agit essentiellement d'éviter d'allouer un processeur à un processus si ce dernier ne peut disposer d'une taille de mémoire au moins égale à son parachore. On est ainsi amené à traiter de façon globale (et non comme deux problèmes distincts) l'allocation des ressources "mémoire" et "processeur". La difficulté principale réside ici dans l'ignorance a priori du comportement des programmes. Dans l'ordre de complexité croissante, deux modes de résolution de ce problème peuvent être envisagés :

- allocation statique : dans l'ignorance du comportement des programmes, on adopte une stratégie invariante, dont les paramètres sont déterminés une fois pour toutes pour un cas "moyen". Une telle solution a le mérite de la simplicité, mais une mauvaise estimation des paramètres (ou, ce qui est plus grave, une dispersion importante des caractéristiques de la charge) peut faire échouer cette politique. On risque en particulier une sous-utilisation des ressources.

- allocation dynamique : on tient compte ici du comportement effectif des programmes pour faire varier les paramètres d'utilisation du système. Les diverses stratégies envisageables se différencient :

- par la nature des variables observées pour caractériser le comportement des programmes et du système dans son ensemble,

- par la nature des paramètres contrôlés.

Nous examinons ici les principales stratégies connues.

3.21 Allocation statique

Dans ce mode d'allocation, on attribue de façon fixe à chacun des programmes en concurrence pour l'usage de la mémoire une fraction fixe de cette mémoire. On garantit ainsi l'indépendance mutuelle des programmes. Lorsque le parachore d'un programme est inférieur à la fraction de mémoire qui lui est impartie, cette mémoire est perdue (elle n'est pas utilisable par les autres programmes). S'il est plus grand, le risque d'écroulement apparaît (risque

multiplié si le dépassement a lieu pour plusieurs programmes). Ce mode d'allocation amène donc à surestimer la place nécessaire. Il est intéressant de donner des indications quantitatives sur ce point.

On examine à cet effet la façon dont réagit un système prévu pour un comportement donné des programmes en présence d'une fluctuation de ce comportement. Cette étude a été faite dans [C2] en utilisant le modèle de comportement local décrit au § 2.323, où la taille du "working set" est une variable aléatoire gaussienne. On considère une mémoire de taille M divisée en n blocs de taille identique b = M/n alloués chacun à un programme de mêmes caractéristiques (m, σ), et on calcule la probabilité f de "non écroulement", condition que l'on traduit par le fait qu'aucun bloc ne se trouve saturé. Cette probabilité s'exprime par

$$f = \left[\Phi\left(\frac{b - m}{\sigma}\right)\right]^n \qquad \text{avec} \quad M = nb$$

où Φ est la fonction de distribution pour une variable gaussienne normalisée (moyenne nulle, variance unité). La figure 3.1 extraite de [C2] indique ce comportement :

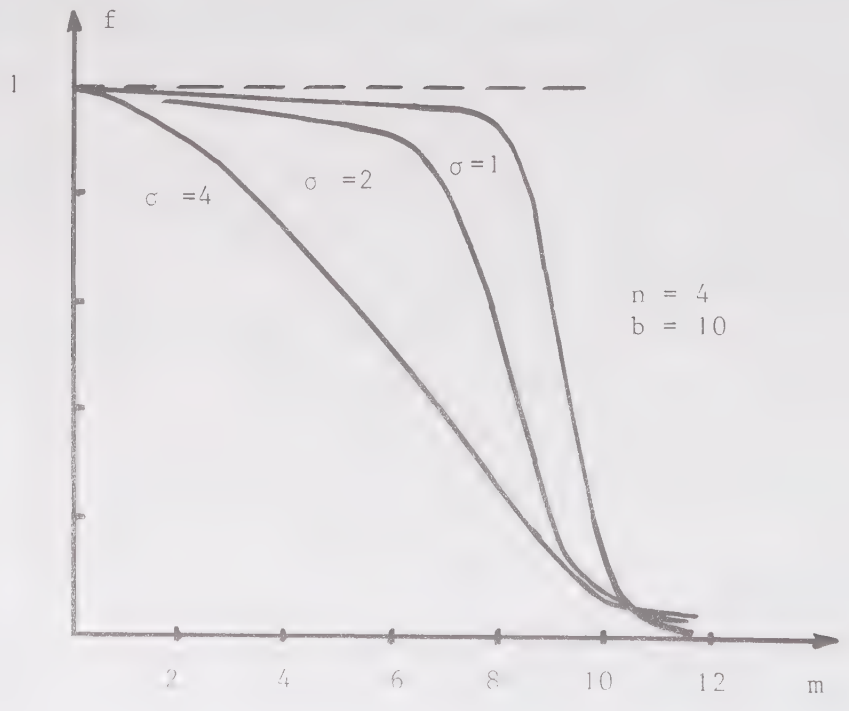


Fig. 3.1 Probabilité de non-écroulement

On constate que l'allocation fixe assure l'absence d'écroulement dans le cas où $\frac{\sigma}{b}$ est assez petit ($\frac{\sigma}{b} < 0{,}3$) et à condition de choisir une taille

de bloc supérieure d'au moins 2 σ à la moyenne du "working set" estimé. Le système d'allocation fixe n'est donc pas recommandable si le comportement des programmes est trop irrégulier (ce qui est conforme à l'intuition), et on peut chiffrer à $\frac{2n\sigma}{M} = \frac{2\sigma}{b}$ la fraction moyenne de mémoire inutilisée. Ces considérations semblent restreindre le type d'allocation statique aux applications où le comportement des programmes est stable et bien connu.

3.22 Allocation dynamique

Les diverses politiques d'allocation dynamique varient suivant la nature des variables contrôlées et les critères utilisés pour la régulation.

L'idée la plus simple est de garder constant un paramètre du système, en faisant varier les autres. Si dans l'exemple du § 3.21, on maintient n constant sans imposer une partition fixe de la mémoire, la condition de non-écroulement étant que la somme des "working sets" des n programmes soit au plus égale à la taille de la mémoire, on obtient pour la probabilité de non-écroulement [C2] :

$$f = \Phi\left(\frac{M - nm}{\sqrt{n}\,\sigma}\right)$$

Cette formule est à comparer avec celle donnée au § 3.121, (avec les mêmes notations). La figure 3.2 ([C2]) permet une comparaison des deux stratégies d'allocation

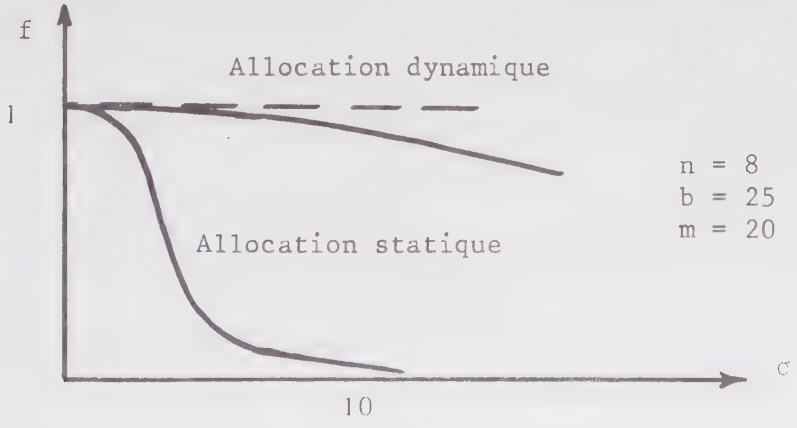


Fig. 3.2

On voit que la stratégie d'allocation dynamique est beaucoup moins sensible que la stratégie statique à une variation des caractéristiques de la

charge.

3.221 Le limiteur de charge

L'étude donnée dans [C2] se fonde sur une population idéale de programmes pour mettre en évidence les caractéristiques comparées de l'allocation statique et dynamique de mémoire. Le dispositif expérimental décrit dans [S3] sous le nom de "limiteur de charge" ("load leveler") est un exemple d'un mécanisme régulateur intégré dans un système détectant la tendance à l'écroulement et agissant sur le système pour éliminer ce danger. Ce dispositif a été implanté sur le système expérimental M44/44X (IBM).

Les réalisateurs du limiteur de charge ont choisi de caractériser la charge du système par les deux paramètres suivants, mesurés sur un intervalle de temps Δt (intervalle d'échantillonnage) qui est un paramètre réglable du système :

- fraction du temps où l'unité centrale est active,
- nombre de remplacements de page effectués durant Δt.

A chaque instant d'échantillonnage, l'état du système peut être défini par les valeurs de ces deux paramètres, durant l'intervalle écoulé, et représenté par un point sur un diagramme (activité de l'UC - nombre de transferts de page). On peut schématiquement distinguer trois zones sur ce diagramme:

- une zone A où la fraction d'activité de l'unité centrale est importante - (c'est la zone de fonctionnement considérée comme "normale",

- une zone C caractérisée par un débit élevé de pages et une faible activité de l'unité centrale. L'interprétation la plus probable de cette situation est que de trop nombreux programmes réclament les services de l'unité centrale, certains d'eux ayant en mémoire un nombre de pages inférieur à la taille critique (parachore). Ces programmes ne peuvent alors bénéficier que de très brefs intervalles d'exécution entre deux défauts de page successifs.

- une zone B, où l'activité de l'unité centrale est faible et les transferts de pages peu nombreux. Le système est alors dans une situation où les ressources sont sous-utilisées par suite d'un trop faible nombre de demandes.

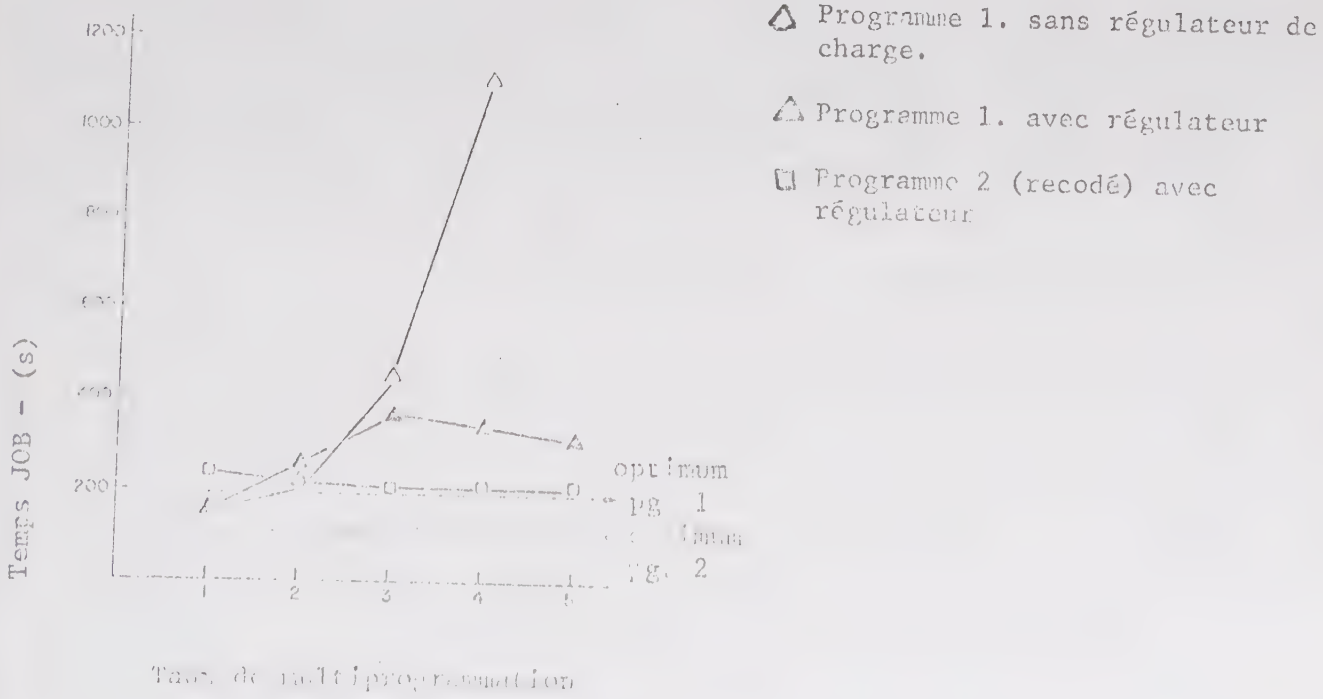
Le principe de fonctionnement du limiteur de charge est de mesurer la charge du système, qui vient d'être définie, et de ramener le système vers l'un des états A ou B lorsqu'il tend à passer dans l'état C. Cela est fait en restreignant le nombre de programmes admis à coexister dans la mémoire et donc à demander les services de l'unité centrale.

Des programmes demandeurs se trouvent ainsi écartés ; il faut toutefois les satisfaire dès que possible, ce qui nécessite de faire un test sur l'existence de programmes en attente dès que le système n'est plus surchargé.

Les résultats obtenus sont spectaculaires, eu égard à la simplicité du mécanisme.

Le but recherché (maintenir le système dans l'état "normal" et éliminer autant que possible l'état de sous-utilisation et surtout l'état de surcharge) est atteint ; toutefois, on constate parfois un certain "pompage" explicable par le caractère rudimentaire de la régulation.

La figure 3.3 extraite de [B6] montre l'amélioration obtenue grâce à l'utilisation du limiteur de charge.



3.222 Expériences de Newcastle

Une étude de simulation a été menée récemment à l'Université de Newcastle-upon-Tyne sur la prévention de l'écroulement dans un système multiprogrammé. [A1] Le comportement simulé des programmes tenait compte des constatations expérimentales : variation non-linéaire de la probabilité de défaut de page avec la taille de la mémoire disponible (existence d'un parachore), et renouvellement assez lent de la population du "working set".

Plusieurs variantes d'un algorithme d'allocation de mémoire ont été simulées. Ces algorithmes sont dénommés "implicites" par leurs auteurs car ils n'agissent pas directement sur le degré de multiprogrammation.

L'idée de base de l'algorithme est d'introduire parmi les processus demandeurs de mémoire un ordre fixe de priorité. La mémoire est allouée librement (par pages) aux demandeurs jusqu'à épuisement. Lorsqu'il n'y a plus de pages libres, on procède ainsi : s'il existe des processus détenteurs de pages, de priorité inférieure à celle du demandeur, une page est retirée au moins prioritaire de ces processus au profit du demandeur ; sinon, le demandeur reste en attente, et sa requête ne sera satisfaite qu'après libération de la mémoire par un processus plus prioritaire que lui.

On constate ainsi que le processus le plus prioritaire bénéficie du même service que s'il était seul dans le système, la qualité du service diminuant avec la priorité. On constate également qu'il se produit un ajustement implicite du degré de multiprogrammation en fonction de la charge : un processus de faible priorité ne peut s'exécuter que s'il y a des pages libres et sera éliminé progressivement si des demandeurs plus prioritaires multiplient leurs demandes.

Cet algorithme donne les résultats escomptés (meilleure utilisation des ressources mémoire et processeur) mais on peut lui reprocher une tendance à l'accumulation de pages par les processus à forte priorité : même si elles ne sont pas réutilisées, les pages resteront en mémoire. Il est intéressant de constater qu'une modification de l'algorithme, tendant à reprendre des pages aux programmes qui en possèdent le plus, conduit à l'écroulement du système : on tend vers une situation où la mémoire est équipartie entre les demandeurs, aucun processus ne pouvant loger son "working set" en mémoire. Plus efficace est une modification consistant à libérer périodiquement (par recopie de son contenu en mémoire secondaire) une page appartenant aux processus actifs ou bloqués intrinsèquement (à l'exclusion de ceux en attente de pages).

Remarquons enfin que la méthode utilisée ne semble pas applicable à un système comportant des contraintes de temps de réponse (usagers conversation-

nels). Son domaine est la multiprogrammation de travaux de fond (avec éventuellement un travail d'avant-plan de haute priorité).

3.223 Expériences de RCA

L'intérêt des expériences faites sur le système TSOS de RCA [M1] est de montrer sur un système réel comment l'apparition de l'écroulement peut être prévue et évitée.

La politique d'allocation de mémoire dans le système TSOS consiste en une réservation "lâche" : un processus ne peut recevoir un processeur que si une réservation préalable de mémoire lui a été faite ; mais cette réservation n'a pas un caractère limitatif, et le processus peut acquérir des pages de mémoire au delà du nombre réservé. Un processus bénéficiant d'une réservation de mémoire est dit "actif" dans la terminologie des auteurs, que nous suivons dans ce paragraphe.

La réservation est fondée sur la mesure de la taille du "working set" des processus : la période de mesure choisie est celle de la dernière période d'activité du processus. Cette période est par ailleurs limitée supérieurement pour éviter toute accumulation de pages par un processus. Quant à l'allocation de processeur, elle se fait par balayage cyclique ("round robin") parmi les processus actifs.

Un processus pouvant dépasser le seuil fixé par sa réservation de mémoire, il est possible d'arriver à une situation où un processus demande une page alors que toutes les pages sont attribuées à des processus actifs. Pour résoudre ce cas de blocage, un des processus actifs est alors "désactivé", libérant sa mémoire. L'algorithme de choix de cette victime n'est pas explicité. Le cas où un programme est partagé par plusieurs processus n'est pas non plus mentionné.

Les expériences sur l'apparition de l'écroulement ont consisté, sous une charge standard de programmes produits par un générateur, à mettre hors service le mécanisme de réservation préalable, permettant ainsi à tout processus demandeur d'acquérir des pages de mémoire. Dans ces conditions, on constate effectivement l'écroulement du système lorsque le nombre d'usagers est tel que la somme de leurs "working sets" moyens dépasse la taille de la mémoire disponible. En revanche, ce nombre peut être dépassé lorsque le mécanisme de réservation est en fonction. Les courbes de la figure 3.4 extraites de [M1] illustrent cette situation.

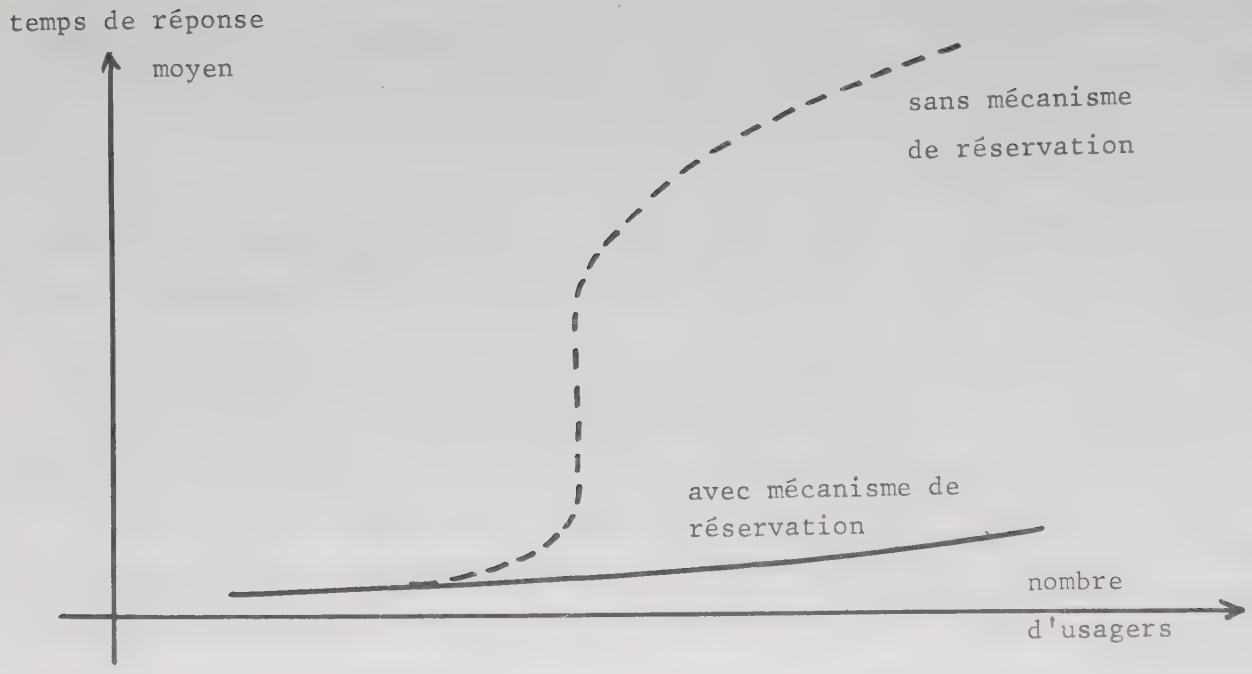


Fig. 3.4 : Simulation de l'écroulement d'un système

3.3 Conclusion

La mise en évidence de certaines caractéristiques constantes dans le comportement des programmes a déterminé une évolution dans la conception de l'allocation de la mémoire et du processeur : l'influence des algorithmes de remplacement de pages a été reconnue secondaire vis à vis des risques d'écroulement provoqués par une allocation incontrôlée de la mémoire centrale. Si les causes du phénomène d'écroulement commencent à être analysées (au point de pouvoir le déclencher de manière prévisible, cf [M1]), les recherches sont moins avancées en ce qui concerne les remèdes.

Toutes les méthodes que nous avons examinées présentent la caractéristique commune d'ajuster dynamiquement le dégré de multiprogrammation en fonction de la charge ; mais les modalités de cet ajustement sont très diverses : action explicite sur le nombre de programmes en mémoire, réservation préalable de mémoire, ajustement "implicite" par croissance ou disparition des programmes. De même, les critères sur lesquels se fonde la régulation sont divers : charge du système, comportement passé des programmes, priorités imposées de l'extérieur.

En outre, plusieurs questions restent ouvertes :

- comment estimer, pour un programme donné, la taille de sa mémoire utile ?
- comment tenir compte des programmes partagés ?
- dans quelles conditions doit-on employer le préchargement ?

- que faire en cas de franchissement des limites prévues pour l'accroissement d'un programme ?

Ces problèmes ne semblent pas, dans l'état actuel des recherches, traitables de façon générale par une étude théorique. Une étude expérimentale en sera donnée, à partir des solutions adoptées pour le système ESOPE, dans la troisième partie de ce travail.

Deuxième Partie

Le Système ESOPE

Chapitre 4 - Objectifs et description générale du Système ESOPE

Chapitre 5 - Gestion des processus et accès à l'information

Chapitre 6 - Conception et réalisation de l'allocateur

Chapitre 7 - Organisation du système en processus parallèles.

N.B. Dans cette partie, un certain nombre de mécanismes du système ESOPE sont décrits dans une notation algorithmique. L'explication de la notation et la liste des variables utilisées sont données en annexe.

Créer n'est pas difficile, c'est décomposer qui est difficile

Alain
Propos

Chapitre 4

OBJECTIFS ET DESCRIPTION GENERALE DU SYSTEME ESOPE

4.1 Objectifs et spécifications générales du projet

La décision, prise en 1968, de réaliser à l'IRIA un système d'exploitation en temps partagé sur le calculateur CII 10070, répondait à un double impératif :

- développer des recherches sur la méthodologie de conception et de réalisation des systèmes d'exploitation à partage de ressources, et démontrer la validité des principes établis par la réalisation effective d'un système opérationnel, et une évaluation précise de son comportement,
- pourvoir l'IRIA d'un outil de travail conversationnel propre à satisfaire les besoins d'un ensemble d'usagers, soumettant des travaux couvrant divers domaines de l'informatique de base ou appliquée, avec prédominance de la mise au point sur l'exploitation (caractéristique de travaux de recherche).

Les spécifications externes du système ont été définies en fonction du second objectif, compte tenu du matériel disponible.

Le système ESOPE est destiné à gérer trois types de travaux, classés ci-dessous par ordre de priorité décroissante :

- un travail dit "en temps réel", caractérisé par les deux propriétés suivantes :
  - intervention explicite du temps physique (matérialisée par des interruptions d'horloge), à des instants dépendant du problème en cours d'exécution,
  - possibilité de gérer simultanément plusieurs terminaux conversationnels, en permettant la communication entre ces terminaux sous le contrôle du sous-système "temps réel".
- un certain nombre de travaux conversationnels indépendants exécutés à partir de terminaux à distance (téléimprimeurs ou consoles alphanumériques).
- un travail de fond, constitué par un train de programmes traités en mono-

programmation. Ce travail est chargé à partir d'un lecteur de cartes.

Parmi les applications prévues pour le travail "en temps réel", citons (liste non limitative) :

- les jeux d'entreprise
- les travaux d'enseignement programmé
- la simulation de système de gestion en temps réel.

Par ailleurs, et ceci est une autre caractéristique originale d'ESOPE, le système permet l'utilisation d'une console graphique. Celle-ci, directement reliée à l'un des blocs de la mémoire principale, est destinée à des recherches sur les langages graphiques de communication et les structures de données correspondantes. En ce qui concerne la priorité des travaux exécutés sur la console graphique, on distingue deux cas :

- les interactions nécessitant une réponse rapide pour des raisons physiques (gestion du photostyle, par exemple) sont traitées en forte priorité.
- les autres types d'interaction sont traités avec la même priorité que les usagers conversationnels ordinaires.

Il est prévu que le travail interactif à partir des terminaux conversationnels constituera l'essentiel de l'activité du système. Les clients en conversation avec le système sont indépendants, mais peuvent échanger de l'information par l'intermédiaire d'un ensemble de fichiers mis à leur disposition. Par ailleurs, un programme mis au point de façon interactive peut être exécuté comme travail de fond. La communication se fait également dans ce cas par le système de gestion de fichiers.

4.2 Caractéristiques du matériel

La configuration de l'ordinateur CII 10070 sur lequel est implanté le système ESOPE est schématisée sur la figure 4.1.

Les ressources disponibles sont :

- une unité centrale, pourvue de toutes les options (virgule flottante, opérations décimales, trois groupes d'interruption externes, quatre horloges),
- deux jeux supplémentaires de 16 registres,
- une mémoire principale comprenant 8 blocs de 16 K mots de 32 bits (l'un des blocs est pourvu d'une voie d'accès supplémentaire pour la console graphique). Un dispositif clé-verrou assure une protection d'écriture par page de 512 mots,

- un dispositif topographique assurant une pagination à un niveau :
Une mémoire virtuelle de 256 pages de 512 mots est implantée, la correspondance page virtuelle-page physique se faisant par 256 registres de 8 bits indiquant pour chaque page virtuelle un numéro de page physique associé.
La protection d'accès à chaque page virtuelle est associé par 2 bits,

- deux unités d'échange multiples à 400 K-octets/s, fonctionnant en parallèle avec l'unité centrale,

- un ensemble de 3 disques à têtes fixes (RAD), avec une seule unité de liaison (3 x 3 M. octets, durée d'un tour : 34 ms, temps de transfert d'une page : 12 ms),

- une mémoire de masse constituée par 8 disques à têtes mobiles (DIMAS) (8 x 25 M. octets),

- des périphériques lents : 1 lecteur de cartes 1200 mm, 2 imprimantes 600 l/mn, 6 dérouleurs de bandes magnétiques 60 K. octets/s.

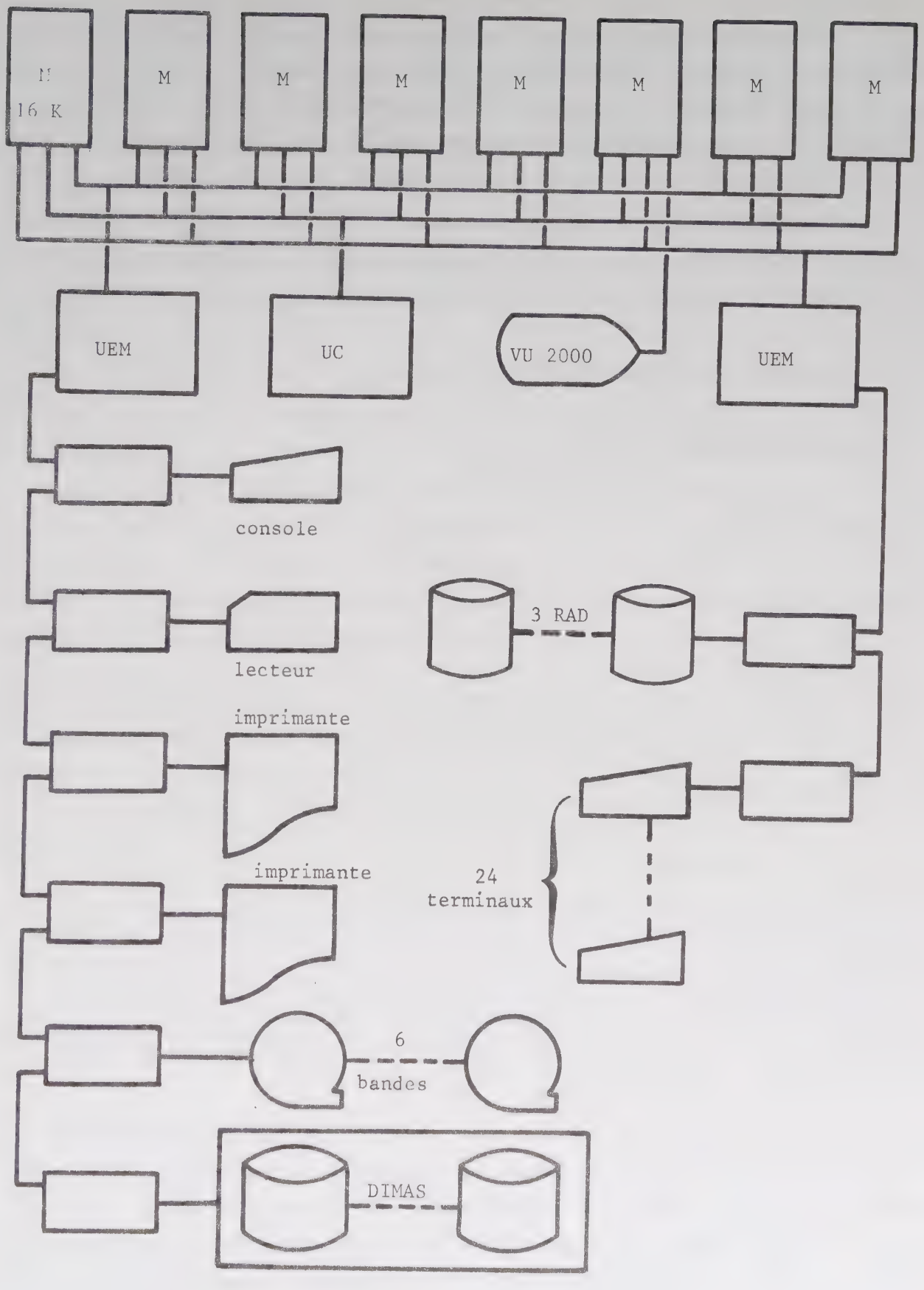


Fig. 4.1 : Configuration du système 10070 de l'IRIA

L'ensemble des terminaux conversationnels comprend 16 téléimprimeurs SAGEM 8/200 à 150 bauds, fonctionnant en mode "full-duplex" (clavier logiquement distinct de l'imprimante, permettant l'émission et la réception simultanée de caractères, et 8 terminaux alphanumériques SYNELEC. Les deux types de terminaux sont entièrement interchangeables, et gérés par le même programme.

Les particularités du matériel ayant influencé de façon déterminante la conception ou la réalisation des diverses parties du système seront décrites plus en détail lors des discussions correspondantes.

4.3 Principes de conception et de réalisation

4.31 Les notions fondamentales

Un objectif important du projet étant de contribuer au développement des méthodes de conception et de réalisation de système, on a cherché à extraire de l'"état de l'art" actuel un petit nombre de concepts, et à construire aussi logiquement que possible le système à partir de ces concepts. Compte tenu de ces objectifs, on s'est donc efforcé (sans toujours y parvenir) d'écarter autant que possible des spécifications externes du système tout ce qui constituerait "une augmentation purement quantitative de la somme de travail à faire" [D6]. De plus, on a essayé de préserver aux mieux, au cours des étapes de la réalisation, la structure définie lors de la conception.

L'analyse initiale a permis de dégager trois classes de problèmes :

- la gestion des activités parallèles, aussi bien des utilisateurs que du système lui-même, pour lesquelles on a utilisé la notion de processus. Des considérations d'homogénéité ont conduit à réserver un traitement uniforme à tous les processus, appartenant ou non au moniteur et à choisir un moyen unique de synchronisation, les sémaphores. On a enfin introduit la notion d'usager considéré comme classe de processus coopérants, la coopération programmée étant impossible entre usagers de processus différents.

- les problèmes de gestion de l'information, qui comprennent :
  - la définition d'un mode de repérage de l'information, permettant aux utilisateurs d'accéder aux objets qu'ils définissent.
  - la protection de l'information contre des actions intempestives, ainsi que la garantie du secret dans les cas où il est requis.
  - la possibilité de partager des informations entre plusieurs utilisateurs sans avoir à en constituer plusieurs exemplaires physiquement distincts.

Ce problème d'accès à l'information a été résolu en définissant un schéma d'adressage utilisant au mieux le matériel disponible, tout en permettant un accès commode à l'information. Un ensemble de dispositifs câblés et programmés simule ainsi sur la machine réelle un mode d'accès à l'information permettant d'atteindre à travers une mémoire virtuelle le triplet <client, fichier, article>,

La liaison entre les processus et les informations qu'ils manipulent peut maintenant être faite : des considérations qui seront développées en détail au chapitre 5 ont conduit à associer l'usager, défini plus haut, à la mémoire virtuelle.

- l'allocation de ressources, en conséquence des choix précédents, se fait donc aux usagers. Toujours par souci d'uniformité de conception, il a été décidé d'utiliser pour l'allocation de ressources les mécanismes existant pour la synchronisation des processus, ce qui entraine deux conséquences :

  - traitement des incidents technologiques par des sémaphores spéciaux,

  - décomposition de l'allocateur en processus, communiquant avec ceux des usagers par les sémaphores ainsi introduits.

En ce qui concerne les algorithmes proprement dits de l'allocation de ressources, ils ont été inspirés par les considérations développées à la fin du chapitre 3, c'est-à-dire :

- concevoir de façon globale la gestion du processeur, de la mémoire centrale, et de la mémoire secondaire.
- tenir compte du comportement observé des usagers, pour compenser le manque d'information a priori sur ce comportement.

Avant d'exposer de façon plus détaillée, dans les chapitres qui suivent, la structure du système, il ne paraît pas inutile de donner quelques précisions sur les techniques utilisées pour sa réalisation.

4.32 Techniques de réalisation

Le système a été écrit dans le langage LP 10070. C'est un langage inspiré du PL 360 [W3], adapté à la structure du calculateur 10070. Les diverses unités adressables sur cette machine (y compris les registres généraux) correspondent aux types des variables manipulées par le langage. Tout en permettant un contrôle strict du code objet produit, le langage LP 10070 offre des instructions de contrôle de l'exécution qui l'apparentent à un langage de haut niveau, avec les avantages correspondants (facilité d'écriture, de lecture,

de transmission, de modification, structure du programme bien apparente).

Le système a été réalisé sous forme de plusieurs versions successives. On s'est efforcé de respecter la structure générale du système (en particulier la décomposition en processus) dès la version la plus élémentaire, les divers processus étant enrichis d'une version à l'autre, mais leur nombre et leurs relations mutuelles restant inchangés.

La première version du système était destinée à vérifier l'exactitude de la construction logique en processus et le fonctionnement des mécanismes de synchronisation. Les principales ressources étaient attribuées statiquement aux usagers. L'allocation dynamique des ressources fut ensuite progressivement implantée au cours des versions suivantes : utilisation d'une mémoire topographique chargée à l'avance pour accéder à des pages physiques statiquement allouées, puis allocation dynamique des pages physiques, couplage dynamique des pages virtuelles, et enfin allocation dynamique d'espace sur disque pour la création de fichiers.

Chapitre 5

GESTION DES PROCESSUS ET ACCES A L'INFORMATION

5.1 Gestion des processus

Avant d'exposer en détail le mécanisme de gestion des processus choisi pour le système ESOPE, nous allons examiner brièvement les caractéristiques des principales techniques existantes pour la communication programmée entre processus.

5.11 Mécanismes de synchronisation entre processus

Le terme de synchronisation que nous utiliserons pour nous conformer à l'usage établi, n'est pas entièrement approprié ici puisqu'il évoque l'idée de temps, qui est précisément absente dans ce contexte, tout au moins en tant que mesure de durée. Le déroulement d'un processus est considéré comme la succession des états (P_i, M_j) de la machine virtuelle associée à ce processus, machine composée d'une mémoire (états M_i) et d'un processeur (états P_i), la vitesse d'évolution entre ces états n'entrant pas en considération.

Rappelons qu'un processus peut lui-même se trouver dans deux états : actif ou bloqué ; l'état bloqué est défini comme un état n'ayant pas de sucesseur. Le problème de l'interaction entre processus consiste à construire un mécanisme permettant à un processus actif (soit A)

- de passer à l'état bloqué, en attendant un signal d'un autre processus,
- d'activer un autre processus (scit B), en lui transmettant éventuellement de l'information.

Remarquons que, dans le second cas, le processus activé B peut déjà se trouver à l'état actif. On doit donc définir de façon plus complète l'effet de l'opération d'"activation" lorsqu'on se trouve dans cette circonstance. Il y a deux possibilités :

a) le signal d'activation est perdu. Il n'a aucun effet immédiat et n'est pas mémorisé,

b) le signal d'activation est mémorisé. Son effet sera de maintenir le processus B actif lorsque celui-ci exécutera sa première opération de blocage.

Un second choix concerne le caractère direct ou indirect du mécanisme d'activation. Dans un mécanisme direct, les actions portent sur des processus identifiés explicitement : l'identité du processus concerné est un paramètre des différentes actions. Dans un mécanisme indirect, au contraire, toutes les actions se font par l'intermédiaire de variables communes auxquelles ont accès tous les processus.

Les deux caractéristiques (appel direct ou indirect, mémorisation ou non) peuvent être combinées de diverses manières. A titre d'exemple, nous présenterons deux mécanismes, l'un direct avec mémorisation partielle, l'autre indirect sans mémorisation. On trouvera dans [F1] une discussions plus complète des mécanismes d'interaction entre processus.

5.111 Le "wakeup waiting switch" ([S1],[L1])

C'est un mécanisme direct, avec mémorisation. A chaque processus est associé un indicateur binaire ("wakeup waiting switch", ou "wws"), pouvant prendre les valeurs 0 et 1. Deux opérations fondamentales sont possibles sur ces indicateurs :

block (q)
wakeup (q)

où q désigne le processus auquel appartient le "wws" concerné.

L'algorithme de ces opérations peut s'exprimer comme suit :

block (q) exécuté par le processus q :

```
begin
    if wws (q) = 0 then
       begin
       état (q) := bloqué ;
       blocage du processus q ;
       end
    else wws (q) := 0
end
```

block (q) exécuté par le processus p n'a aucune action sur le processus appelant p mais oblige le processus q à exécuter block (q) suivant l'algorithme donné plus haut.

wakeup (q) exécuté par le processus p :

```
begin
if état (q) = bloqué then
     begin
     état (q) := actif ;
     activation du processus q
     end
else wws (q) := 1
end
```

On voit que le "wws" d'un processus peut mémoriser une (et une seule) activation éventuelle d'un processus alors que celui-ci est encore actif. La valeur du "wws" d'un processus actif représente un "potentiel d'activation" : si cette valeur est 0, le processus sera bloqué à la première opération "block" ; sinon; il restera actif. On peut remarquer qu'un ensemble de n activations pendant une période d'activité (n > 1) est équivalent à une activation unique ; c'est là une limitation du mécanisme.

5.112 Les "événements" (LTR)

L'événement du langage LTR est un mécanisme indirect, sans mémorisation. Un événement doit faire l'objet d'une déclaration qui fixe sa portée en tant qu'objet du langage (domaine d'existence). Un événement peut être "attendu" (WAIT) ou "déclenché" (ACTIVATE).

Soit A un événement déclaré. L'instruction WAIT (A) dans le processus p produit le blocage de ce processus jusqu'à l'occurence d'une instruction ACTIVATE (A) dans un autre processus ; mais si ACTIVATE se produit avant WAIT (si un événement est déclenché alors qu'il n'est pas attendu), son effet n'est pas mémorisé, et aucun potentiel d'activation ne lui est attaché. Indiquons que l'opération WAIT peut être associé à des entités plus complexes qu'un simple événement (par exemple : occurence conjointe de deux événements, ou occurence de l'un ou l'autre de deux événements).

Nous allons maintenant décrire en détail le mécanisme adopté pour le système ESOPE.

5.12 Sémaphores

Le mécanisme d'interaction par sémaphores est dû à DIJKSTRA [D5], [D6]. Il est du type indirect avec mémorisation.

5.121 Définition et propriétés fondamentales

Un sémaphore est défini comme l'ensemble d'une variable s à valeurs entières et d'un ensemble Q(s) de processus. Un processus p peut exécuter sur un sémaphore s deux opérations dites primitives auxquelles nous conservons les noms P et V donnés par DIJKSTRA. Aucune modification de s ou Q n'est possible autrement qu'à travers ces primitives.

Nous décrivons ci-dessous l'algorithme de P(s) et V(s). On désigne par p le processus exécutant la primitive.

```
P(s) :  begin
        s := s - 1 ;
        if s < 0 then
                begin
                état (p) := bloqué ;
                faire entrer p dans Q(s) ;
                end
        end

V(s) :  begin
        s := s + 1 ;
        if s ⩽ 0 then
                begin
                choisir un processus q dans Q(s) et l'en extraire ;
                état (q) := actif ;
                end
        end
```

La propriété essentielle des opérations P et V est leur indivisibilité. Chacune de ces opérations est considérée comme un tout, en ce sens que si deux processus tentent d'exécuter "simultanément" deux opérations primitives (soit respectivement p1 exécutant P1 et p2 exécutant P2), ces opérations se dérouleront dans l'ordre P1; P2 ou P2; P1. (en d'autres termes, les primitives ne peuvent être décomposées en actions plus élémentaires susceptibles de s'entremêler chronologiquement). C'est cette propriété qui permet d'éviter les difficultés rencontrées lorsqu'on essaie de synchroniser des processus parallèles par

l'intermédiaire de variables booléennes ; cf à ce sujet la discussion dans [D5].

Nous allons maintenant établir deux propriétés fondamentales des sémaphores. Nous considérons un sémaphore s, et un ensemble de processus susceptibles d'agir sur le sémaphore au moyen des primitives P et V.

Nous imposons les conditions initiales suivantes à l'instant t = 0

$$\begin{cases} s = s_o & (s_o \geqslant 0) \\ Q(s) = \emptyset & \text{(ensemble vide)} \end{cases}$$

Les propriétés énoncées sont vraies à tout instant $t \geqslant 0$.

Propriété 1. Soient respectivement N_p et N_v le nombre total d'opération P et V exécutées par les divers processus sur le sémaphore s entre les instants 0 et t. La valeur du sémaphore à l'instant t est $s_o + N_v - N_p$.

Cette propriété résulte directement de l'indivisibilité des opérations P et V, et du fait que P(s) et V(s) ajoutent respectivement -1 et +1 à la valeur du sémaphore s.

Propriété 2. Soit n le nombre de processus bloqués dans Q(s). On a la relation :

$$n = \underline{\text{if}}\ s < 0\ \underline{\text{then}}\ -s\ \underline{\text{else}}\ 0$$

Cette relation est vraie à l'instant initial. Montrons qu'elle reste vérifiée après l'exécution de P(s) ou de V(s). Nous distinguerons deux cas :

- $s > 0$: L'exécution de P(s) ou de V(s) ne modifie pas la composition de Q(s). La valeur de n reste donc 0.

- $s \leqslant 0$:

- si un processus exécute P(s), il fait s := s - 1. La valeur résultante de s étant négative, le processus se bloque et entre dans Q(s) ; on aura donc n := n + 1 et la relation n = - s restera vérifiée,

- si un processus exécute V(s), il fait s := s + 1. Si la valeur résultante de s est négative ou nulle, alors un processus sortira de Q(s). On aura donc n := n - 1, ce qui conserve la relation n = - s.
Si la valeur résultante est positive, n reste inchangé ; mais dans ce cas s (donc n) valait 0 avant le V(s), et n reste donc égal à 0.

L'expérience de l'utilisation des sémaphores dans la construction du système ESOPE a mis en évidence l'utilité d'un mécanisme rendant compte de la notion d'"événement non mémorisé" (au sens, par exemple de LTR, cf 6.112). Cette

notion a été introduite, par souci d'uniformité, sous la forme d'une primitive supplémentaire D agissant sur un sémaphore et s'ajoutant aux primitives P et V : L'algorithme de D est le suivant :

```
D(s) :  begin
        if s < 0 then
             begin
             activer tous les processus de Q(s) ;
             s := 0
             end
        end
```

Comme P et V, D doit être considéré comme une opération indivisible.

Remarquons que l'introduction de D(s) conserve la propriété 2 (en effet l'état après l'exécution de s est s = 0, Q(s) = ∅.)

L'effet de la primitive D(s) peut s'interpréter de deux façons :

- comme un événement non mémorisé vis à vis des processus éventuellement bloqués derrière le sémaphore s. (si aucun processus n'est bloqué, l'événement ne laisse pas de traces).

- comme l'activation simultanée de tous les processus bloqués derrière s. En ce sens, on peut considérer D comme une opération V répétée, qui peut s'écrire :

```
while s < 0 do V(s)
```

On trouvera une discussion plus détaillée de D dans [F1] et [K3].

5.122 Utilisation des sémaphores

En tant qu'outil de communication entre processus, les sémaphores peuvent être utilisés de deux façons distinctes (cf. [D6]).

- Sémaphores d'exclusion mutuelle

Un sémaphore d'exclusion mutuelle sert à limiter à un nombre fixé à l'avance le nombre de processus susceptibles d'exécuter simultanément une certaine partie de leur algorithme (section critique). Un exemple classique est celui de l'accès à des données communes, dont l'usage est restreint à un processus à la fois.

L'algorithme des divers processus s'écrit alors, introduisant un sémaphore mutex dont la valeur initiale est 1 :

```
processus 1 : begin                    --- processus i : begin    ---
                :                                        :
              P(mutex) ;                               P(mutex)
              section critique                         section critique du
              du processus 1 ;                         processus i ;
              V(mutex) ;                               V(mutex)
                :                                        :
              end                                      end
```

Cet algorithme garantit qu'un processus au plus est dans sa section critique à un instant donné. Supposons en effet qu'il y en ait n, n > 1, à un instant t. Alors, à cet instant, il y aurait eu n opérations P exécutées sur le sémaphore mutex. D'après la propriété 1, la valeur de mutex serait donc 1-n ; d'après la propriété 2, n-1 des processus considérés seraient bloqués, ce qui est en contradiction avec l'hypothèse qu'ils sont dans leur section critique.

De manière plus générale, la valeur k initialement affectée au sémaphore mutex fixe le nombre maximum de processus admis à exécuter simultanément leur section critique (la démonstration du fonctionnement correct de l'algorithme est la même que pour k = 1)

- Sémaphores privés

Dans de nombreuses applications, en particulier en programmation de systèmes, un processus est activé de façon répétitive pour exécuter à chaque appel les mêmes actions. Il est commode dans ce cas de considérer le processus comme cyclique, d'après le schéma suivant :

```
processus p : begin
   REPOS    : P(sempriv)  ;
              algorithme du processus ;
              goto REPOS
              end
```

Pour assurer le fonctionnement correct de ce dispositif, on doit respecter les règles suivantes :

- seul le processus propriétaire peut exécuter l'opération P sur le sémaphore sempriv,
- la valeur initiale de sempriv doit être égale à 0.

Dans sa "position de repos", le processus p est bloqué dérrière son

sémaphore privé. Il est activé par une opération V (ou D) de la part d'un autre processus ; il exécute alors son algorithme (éventuellement plusieurs fois, s'il y a eu des activations multiples) avant de revenir à sa position de repos. Remarquons qu'on a toujours la relation

$$sempriv \geqslant -1 \quad \text{(égalité dans le cas de la position de repos)}$$

On voit que l'emploi des sémaphores d'exclusion mutuelle et des sémaphores privés correspond à deux utilisations logiquement très distinctes. Cette constatation a d'ailleurs conduit plusieurs auteurs([H2], [H3]) à proposer de séparer totalement ces deux utilisations en introduisant deux mécanismes (et notations) distincts.

5.123 Un exemple de synchronisation par sémaphores

Dans le but d'illustrer l'utilisation des sémaphores pour l'exclusion mutuelle et la synchronisation de processus parallèles, nous donnerons ici un exemple tiré du système de gestion de fichiers d'ESOPE. Ce système n'étant pas décrit dans le présent travail, nous présentons d'abord un énoncé schématique du problème, simplifié pour les besoins de l'exposé.

Tout client autorisé à utiliser le système ESOPE possède un catalogue, qui répertorie les noms et divers attributs de ses fichiers permanents. Pour des raisons d'encombrement, les catalogues sont normalement stockés dans une mémoire à accès lent, que nous appellerons ici mémoire externe. Pour être consulté ou modifié, un catalogue doit être préalablement amené en mémoire rapide (plus exactement dans l'ensemble mémoire rapide-disques, que nous appellerons mémoire interne).
Pour le problème qui nous intéresse ici, il est inutile de préciser les modalités du transfert entre mémoire externe et mémoire interne ; nous considérerons ici ces transferts comme des opérations élémentaires ("rappel" dans le sens externe-interne, "sauvegarde" dans le sens contraire).

Le catalogue d'un client x doit se trouver en mémoire interne dans les cas suivants :

- le client x est en activité dans le système
- un client y au moins a demandé à utiliser les fichiers du client x.

Pour éviter d'encombrer la mémoire interne de catalogues inutiles, le nombre de catalogues simultanément présents en mémoire interne est limité ; de plus, il est décidé que tout catalogue est sauvegardé dès que sa présence en mémoire interne n'est plus nécessaire. On entretient à cet effet :

- une table donnant les numéros des clients dont les catalogues sont

internes.
C'est un tableau CL(I) à N_{max} entrées ; CL(I) = 0 indique que l'emplacement I est libre, CL(I) = X, qu'il est occupé par le catalogue du client X. N_{max} est le nombre maximal de catalogues internes.

- un ensemble de compteurs d'utilisation N(I), indiquant le nombre d'utilisateurs de chaque catalogue interne. Quand le compteur N(I) passe d'une valeur positive à zéro, le catalogue occupant l'emplacement I doit être sauvegardé, et l'emplacement correspondant libéré.

Le problème consiste à écrire les procédures de sauvegarde et de rappel des catalogues. Ces procédures doivent pouvoir être utilisées par un nombre quelconque de processus parallèles, ce qui conduit à formuler les contraintes suivantes :

- toute consultation ou modification des tables N et CL doit faire l'objet d'une exclusion mutuelle, pour garantir la cohérence de ces tables,
- si un même catalogue est rappelé "simultanément" par plusieurs processus, le transfert ne doit avoir lieu qu'une seule fois, mais tous les processus demandeurs doivent être prévenus à la fin du rappel,
- si un même catalogue est "simultanément" demandé par un processus et sauvegardé par un autre, alors :
  - ou bien aucun transfert ne doit être fait
  - ou bien la sauvegarde doit être faite avant le rappel.

Aux contraintes précédentes, d'ordre logique, on adjoint deux contraintes d'ordre pratique : la sauvegarde et le rappel étant des opérations longues, susceptibles de donner lieu à des incidents technologiques, on s'interdit d'inclure ces opérations dans une section critique ; de même, on s'interdit tout blocage d'un processus à l'intérieur d'une section critique.

Solution :

Il est nécessaire de détecter les cas de "transit", c'est-à-dire le cas où un mouvement de catalogue est en cours, dans un sens ou l'autre. Pour cela, on adopte la méthode suivante :

CL(I) est mis à jour avant le rappel et après le sauvetage.

N(I) est mis à jour après le rappel et avant le sauvetage.

On a donc les 3 configurations suivantes :

N(I) / CL(I)	= 0	> 0
= 0	emplacement I libre	
> 0	catalogue CL(I) en transit	catalogue CL(I) présent

On introduit deux sémaphores :

- un sémaphore d'exclusion mutuelle pour l'accès aux tables N et CL
- un sémaphore d'attente derrière lequel se bloqueront les processus demandeurs trouvant le catalogue demandé en transit.

La fin de tout transfert doit donc comporter l'activation de tous les processus attendant la fin de ce transfert ; comme leur nombre est inconnu (on ne peut avoir directement accès à la valeur d'un sémaphore), on introduit un compteur des usagers en attente ; ce compteur est consulté et mis à jour à l'intérieur de sections critiques.

L'algorithme de sauvegarde et de rappel utilise deux procédures :

- Demander Catalogue (X) est appelée par un processus désirant accéder au catalogue du client X.
- Libérer Catalogue (X,Y) est appelé par un processus cessant d'utiliser le catalogue du client X, occupant l'emplacement Y en mémoire interne.

Une procédure booléenne Catalogue Présent (X) est utilisée dans Demander, son effet est de déterminer si le catalogue de X est présent ou non en mémoire interne.

```
boolean procédure Catalogue présent (X, Y) ;
     Catalogue présent := {∃ Y, (CL(Y) = X) & (Y < N_max)} ;

procédure Demander Catalogue (X) ;
     begin
TEST : P(mutex) ;
     if Catalogue présent (X, Y) then
          begin                    /* Y = emplacement de catalogue */
          if N(Y) = 0 then
               begin
               Ntransit := Ntransit + 1 ;
               V(mutex) ;
               P(transcat) ;
               goto TEST ;
               end
          else
               begin
               N(Y) := N(Y) + 1 ;
               V(mutex)
               end
          end
     else
          begin
          chercher emplacement libre ;
          (résultat = emplacement Z, ou échec)
          CL(Z) := X ;
          V(mutex) ;
          rappel du catalogue du client X     /* hors section critique */
          P(mutex) ;
          N(Z) := N(Z) + 1 ;
          while Ntransit > 0 do
               begin
               Ntransit := Ntransit - 1 ;
               V(transit) ;
               end;
          V(mutex)
          end
     end ;
```

```
procédure Libérer Catalogue (X, Y) ;
     begin
     P(mutex) ;
     N(Y) := N(Y) - 1 ;
     if N(Y) > 0 then
          V(mutex)                    /* le catalogue reste interne */
     else
          begin
          V(mutex) ;
          sauvegarde du catalogue de X ;      /* hors section critique */
          P(mutex) ;
          CL(Y) := 0 ;
          while Ntransit > 0 do
               begin
               Ntransit := Ntransit - 1 ;
               V(transit)
               end ;
          V(mutex) ;
          end
     end
```

Remarque : on n'a prévu qu'un sémaphore unique de blocage pour les processus en attente de fin de transit. Plusieurs processus peuvent simultanément attendre la fin du transit de plusieurs catalogues différents et tous sont réveillés à la fin de chaque transit. Comme chacun d'eux recherche alors de nouveau le catalogue qu'il attend, la sélection est automatiquement assurée.

5.13 Implantation des processus dans le système ESOPE

5.131 Classes de processus et de sémaphores

L'analyse de l'activité du système fait apparaitre une partition de l'ensemble des processus en groupes de processus mutuellement indépendants, deux processus appartenant à deux groupes différents n'ayant entre eux aucune relation explicite. Par exemple, deux processus appartenant à deux utilisateurs conversationnels différents sont dans ce cas.

La première définition de l'usager dans le système ESOPE est fondée sur cette propriété d'indépendance : un usager est défini comme un groupe de processus susceptible d'avoir des interactions mutuelles explicites ; l'interaction explicite entre processus de deux usagers différents est impossible.

Nous entendons initialement par interaction l'utilisation des primitives de synchronisation définies au § 5.12. Si l'on exclut de travailler avec une population fixe de processus, on doit au moins ajouter à ces primitives les opérations permettant de créer et de détruire des processus ; ces opérations seront définies plus loin avec précision.

Dans ces conditions, le graphe dont les noeuds sont les processus d'un usager et les arcs les relations de paternité (processus créateur, processus créé) est une forêt dont les racines sont les processus existant avant toute création. Il ne s'est pas avéré nécessaire dans ESOPE de prévoir initialement plus d'un processus, et la forêt se réduit ainsi à une arborescence. La racine de cette arborescence est un processus qui préexiste et survit à tous les autres processus de l'usager, dont il est l'ancêtre commun ; il a reçu pour cette raison le nom de processus premier. Les problèmes d'apparition et de disparition du processus premier seront laissés de côté pour être traités au chapitre 7.

La définition de l'usager implique que les processus d'un même usager aient accès à des sémaphores communs. Ces sémaphores sont rendus inaccessibles aux autres usagers pour des raisons de sécurité ; la notion d'usager est ainsi associée à un domaine de validité de sémaphores.

Pour une raison d'uniformité des mécanismes, on a choisi dans ESOPE d'utiliser le mécanisme des sémaphores pour la gestion des ressources, autrement dit pour l'interaction implicite entre processus. Ce choix implique donc l'existence de deux types de sémaphores :

- sémaphores propres à un usager, utilisés pour la synchronisation explicite des processus de cet usager, et inaccessibles à tout autre processus,

- sémaphores globaux accessibles à tous les usagers (avec des restrictions éventuelles pour des raisons de protection) utilisés pour le partage des ressources entre usagers.

Les processus privilégiés dont l'ensemble constitue le moniteur ont accès aux sémaphores globaux du système.

Pour des raisons que nous ne développerons pas ici, mais que l'on pourra trouver dans [F1] , ch. IV, il a été décidé de limiter à un le nombre de processus fils du processus premier à un instant donné. Ce processus unique est appelé traducteur ; il n'y a aucune restriction sur le nombre de ses descendants (sauf celles introduites par la gestion des tables, cf. § 5.132)

On aboutit donc au schéma de la figure 5.1

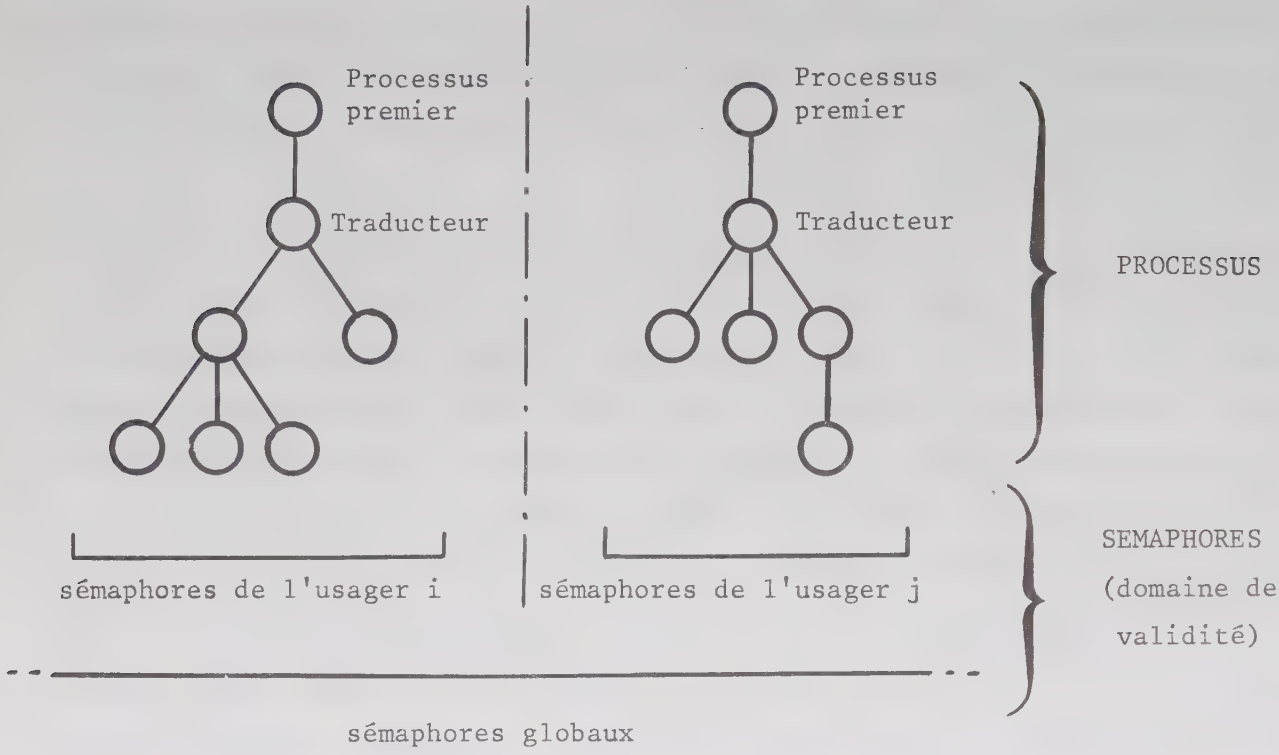


Fig. 5.1 : Processus des usagers

5.132 Structures de données associées aux processus

Les caractéristiques associées à chaque processus du système sont rangées dans des tables. Celles-ci, pour des raisons de simplicité, sont résidentes en mémoire physique. Leur dimension est donc limitée, ainsi que le nombre total de processus du système.
Le nombre de processus du moniteur est fixe ; les processus créés par les usagers sont gérés par l'algorithme du banquier, qui permet d'éviter le blocage pour défaut de place dans les tables de processus. La même organisation est adoptée pour les tables de sémaphores.

L'accès aux tables de processus et de sémaphores doit tenir compte des domaines de validité indiqués au paragraphe précédent. Toutefois, pour des raisons d'homogénéité, la gestion de tous les types de processus et sémaphores est commune. L'accès est contrôlé par le mécanisme suivant :

- tout processus ou sémaphore possède un indice qui est son numéro d'ordre dans la table générale des processus ou des sémaphores,

- les processus et sémaphores créés par les usagers possèdent en outre un nom ; la portée d'un nom est limitée à l'usager qui l'emploie.

Le mécanisme de correspondance entre nom et indice n'est pas accessible aux

usagers.

Les primitives exécutant des actions sur les processus et sémaphores sont donc interprétées différemment suivant le type de processus qui les exécute. S'il s'agit d'un processus du moniteur, ou d'un processus premier d'un usager, le paramètre transmis est interprété comme un indice. S'il s'agit d'un autre processus (c'est alors un processus ordinaire d'un usager) le paramètre transmis est interprété comme un nom interne à cet usager. Le mécanisme de transformation des noms en indices, inclus dans les primitives, assure la protection mutuelle des usagers et la protection du moniteur vis à vis des usagers.

Chaque entrée de la table des processus contient les informations suivantes :

- vecteur d'état du processus : celui-ci comprend :
  - la valeur des 16 registres généraux utilisés par le processus
  - le double mot d'état de programme (PSD)
  - le vecteur d'autorisation, définissant les droits attachés au processus (cf § 5.43)
- la priorité du processus (relativement aux autres processus du même usager,
- le nom du processus (si celui-ci n'est pas un processus du moniteur,
- l'indication de la file de sémaphores où se trouve le processus. Un processus bloqué est nécessairement dans une file de sémaphore (et une seule). Par souci d'homogénéité, les processus prêts d'un même usager (ou du moniteur) sont également rangés dans une file, qui a la même structure qu'une file de sémaphores. Les opérations d'activation et de blocage se traitent uniformément par un changement de file,
- un lien de chaînage dans la file (au sens ci-dessus) où se trouve le processus,
- un lien de chaînage reliant le processus aux autres processus de l'usager (dans l'ordre de priorité décroissante). Ce lien sert, entre autre, à rechercher un processus de nom donné chez un usager, et à exécuter des actions sur l'ensemble des processus d'un même usager,
- l'indication du numéro d'usager auquel appartient le processus.

De même, chaque entrée de la table des sémaphores contient outre le nom et la valeur du sémaphore un lien vers la file des processus bloqués derrière le sémaphore, et un lien vers le sémaphore suivant de l'usager. Des entrées spéciales de la table des sémaphores représentent , toujours pour des raisons d'homogénéité, les files de processus prêts du moniteur et des différents usagers.

Toutes les files définies ci-dessus sont circulaires, simplement chaînées avec pointeur de queue ; pour chacune des deux tables, une file spéciale contient toutes les entrées libres de la table.

5.133 Implantation des primitives de synchronisation

Trois points sont importants dans la réalisation pratique des primitives de synchronisation :

- l'indivisibilité des primitives,
- la transmission des paramètres par le processus appelant,
- la protection contre une utilisation intempestive.

Examinons rapidement les solutions retenues (on se reportera à [F1] pour une description plus détaillée)

- en monoprocesseur, l'indivisibilité s'obtient simplement en masquant toutes les interruptions pendant l'exécution des primitives (cette méthode serait insuffisante en multiprocesseur),

- le passage des paramètres d'appel se fait par les registres, seule partie de l'espace d'adressage propre à chaque processus. Par ailleurs, un certain nombre de paramètres sont implicites ; ici encore, la machine étant monoprocesseur, l'identité du processus appelant, et de l'usager appelant sont déterminées aisément par des variables communes,

- enfin, le principe de la protection consiste à identifier le processus appelant dès l'appel de la primitive. Les actions permises sont déterminées en fonction des droits de ce processus. Ce mécanisme est décrit plus en détail au § 5.4.

5.2 Accès à l'information

5.21 Les objectifs et les contraintes

La fonction d'accès à l'information dans un système informatique, se présente sous plusieurs aspects dont nous énumérons ci-après les principaux :

- conservation "indéfinie" de l'information confiée par les usagers,
- mécanismes permettant le partage d'information entre utilisateurs,
- possibilité d'évolution dynamique de structures d'information pendant la vie du système, par création, destruction, modification de taille.

L'ensemble de l'information accessible à un usager est défini de façon intrinsèque, indépendamment de son support physique. Pour être utilisable pratiquement, cet ensemble d'information doit recevoir une structure, permettant une identification simple des unités d'information que le constituent. La

structure la plus couramment employée est arborescente. Les feuilles de l'arborescence sont les unités élémentaires d'information (articles), les noeuds sont des collections d'articles (fichiers), dont certains (catalogues) jouent le rôle de repères vis à vis d'autres catalogues ou fichiers. Les idées fondamentales sur la structure et les propriétés de cette organisation de fichiers dans les systèmes à partage de ressources ont été développées dans [D1].

La structure externe des fichiers du système ESOPE a été choisie simple, pour les raisons développées au § 4.31. On s'est limité au minimum nécessaire de deux niveaux de catalogues, et à une structure linéaire uniforme pour tous les fichiers : un fichier est une suite ordonnée d'articles identiques. Le choix de la taille de l'article sera discuté plus loin.

La figure 5.2 schématise l'organisation d'ensemble des fichiers.

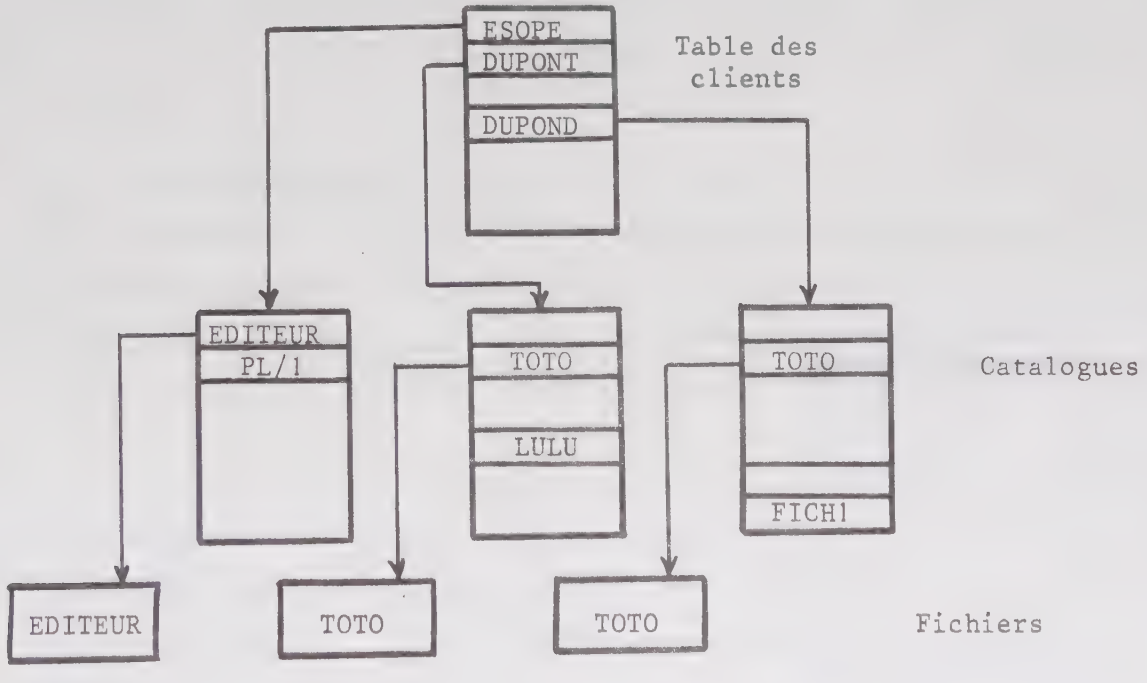


Fig. 5.2 : Organisation externe des fichiers

Un fichier est identifié par son nom précédé du nom du client propriétaire (ex. : ESOPE * EDITEUR, DUPONT * TOTO). Les fichiers communs du système sont catalogués sous le nom d'un client spécial (ESOPE).

L'organisation de l'information étant ainsi définie sous une forme externe, le problème se pose de la réalisation d'une méthode d'accès à cette information compte tenu du matériel disponible, ou, en d'autres termes, de la simulation sur la machine réelle d'une "machine abstraite" réalisant l'accès (lecture ou modification) à l'ensemble des triplets

< nom de client, nom de fichier, numéro d'article >

les trois éléments parcourant l'ensemble des valeurs permises, compte tenu de certaines contraintes encore à préciser (droits d'accès, protection).

Examinons le dispositif d'adressage du calculateur 10070. Ce dispositif appelé mémoire topographique assure une mise en correspondance de deux ensembles, l'ensemble Φ des adresses φ des mots de la mémoire physique ($\varphi \in [0, 2^N-1]$) et l'ensemble V des adresses v obtenues par calcul d'adresse (indirection, indexation). Cet ensemble V porte le nom de mémoire virtuelle ($v \in [0, 2^{17}-1]$).

Le mécanisme de passage de v à φ est schématisé sur la figure 5.3. La correspondance se fait page par page.

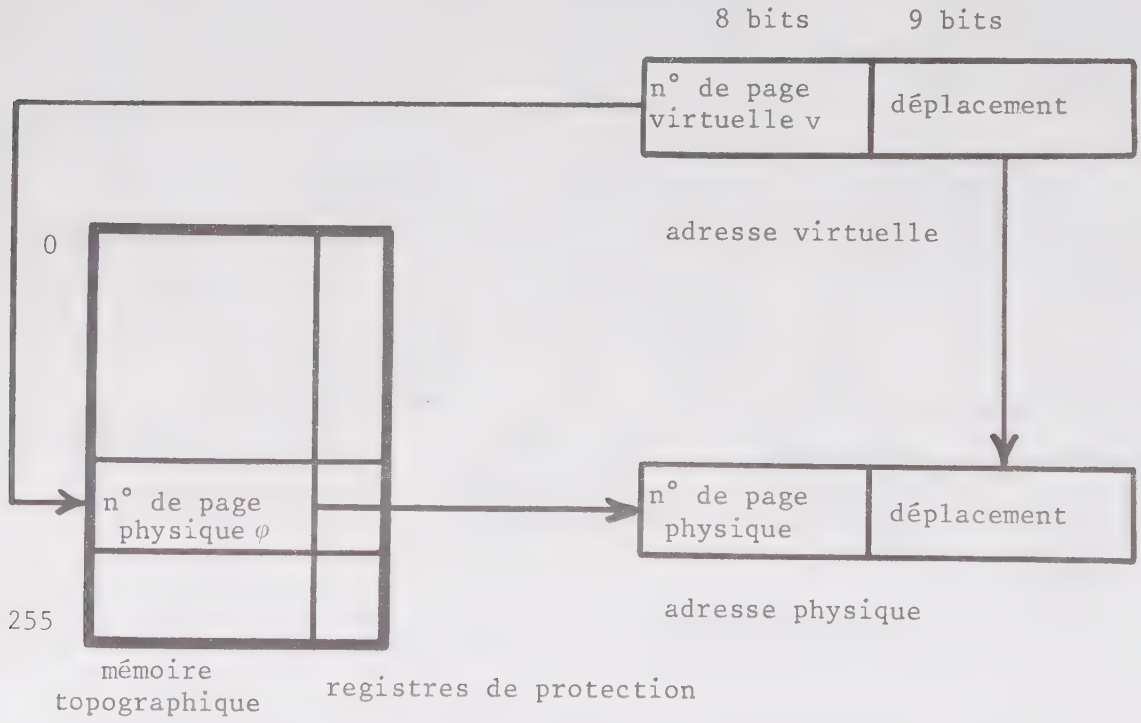


Fig. 5.3 : Réalisation de la mémoire virtuelle

A chaque contenu des registres topographiques correspond ainsi une réalisation particulière de la mémoire virtuelle, que l'on désignera dans la suite sous le nom de feuillet.

5.22 Choix d'un mécanisme

Le problème est maintenant d'assurer l'accès à une information désignée par le triplet <nom de client, nom de fichier, numéro d'article> à l'aide du mécanisme disponible, qui permet l'accès au couple <feuillet, numéro de page virtuelle>.

Remarquons d'abord que, grâce à la structure arborescente définie plus haut, le problème peut être résolu en deux étapes indépendantes :

<nom de client, nom de fichier> ⟶ <repère interne de fichier>

<repère interne de fichier, numéro d'article> ⟶ <feuillet, numéro de page virtuelle>

Nous nous intéresserons d'abord à la seconde étape. Deux solutions extrêmes sont envisageables a priori, la première consistant à associer terme à terme les deux éléments des doublets, la seconde à les disjoindre totalement, en matérialisant la correspondance par un transfert effectif d'information. Examinons successivement ces deux possibilités.

solution 1 : Correspondance biunivoque

On réalise les associations :

<repère interne de fichier> ⟶ <feuillet>

<numéro d'article> ⟶ <numéro de page virtuelle>

Il est clair qu'une telle solution, malgré sa simplicité conceptuelle, ne répond pas aux objectifs fixés, au moins pour les raisons suivantes, qui tiennent essentiellement à la réalisation de la mémoire virtuelle :

- tout changement de fichier se traduit par un rechargement de la mémoire topographique ; or de tels changements sont fréquents, ne serait-ce qu'en raison de la nécessité de séparer les fichiers de programmes exécutables (partageables) et les fichiers de données (privés)

- la taille de tout fichier est limitée à celle de la mémoire virtuelle.

Nous avons toutefois cité cette solution car elle paraît la plus naturelle. Dans le système MULTICS, où le matériel a été conçu en même temps que le système d'exploitation, c'est en fait une solution de ce type qui a été adoptée (les "segments" au sens de MULTICS remplaçant les "feuillets"). Les deux inconvénients mentionnés ci-dessus sont éliminés :

- le mécanisme d'adressage permet le passage rapide d'un segment à l'autre,

- la taille des segments est suffisamment importante pour être

considérée comme infinie pour tous besoins pratiques.

solution 2 : Transfert explicite

On considère maintenant un feuillet comme un espace de travail, où l'information est accessible et modifiable par un processeur. L'accès à un élément <repère de fichier, numéro d'article> est réalisé en trois phases :

- entrée : transférer l'élément $<f,x>$ à un emplacement $<u,v>$ (ou une suite d'emplacements $<u, v_i>$ si la taille de l'article est supérieure à une page),
- accès : opération en mémoire virtuelle, sur l'élément $<u,v>$
- sortie : transférer l'élément $<u,v>$ à l'emplacement $<f,x>$

(cette dernière opération n'est nécessaire que s'il y a eu modification de l'élément).

Examinons à présent les opérations physiques qu'implique un tel mécanisme. L'information constituant le contenu des différents feuillets est rangée dans la mémoire secondaire, amenée en mémoire physique par le mécanisme du défaut de page, et éventuellement renvoyée en mémoire secondaire lorsque la page physique est réutilisée. La zone de mémoire secondaire réservée à la conservation du contenu des divers feuillets est désignée en général sous le nom de mémoire d'instance.

Le trajet physique d'un article de fichier est schématisé sur la figure 5.4 ci-dessous :

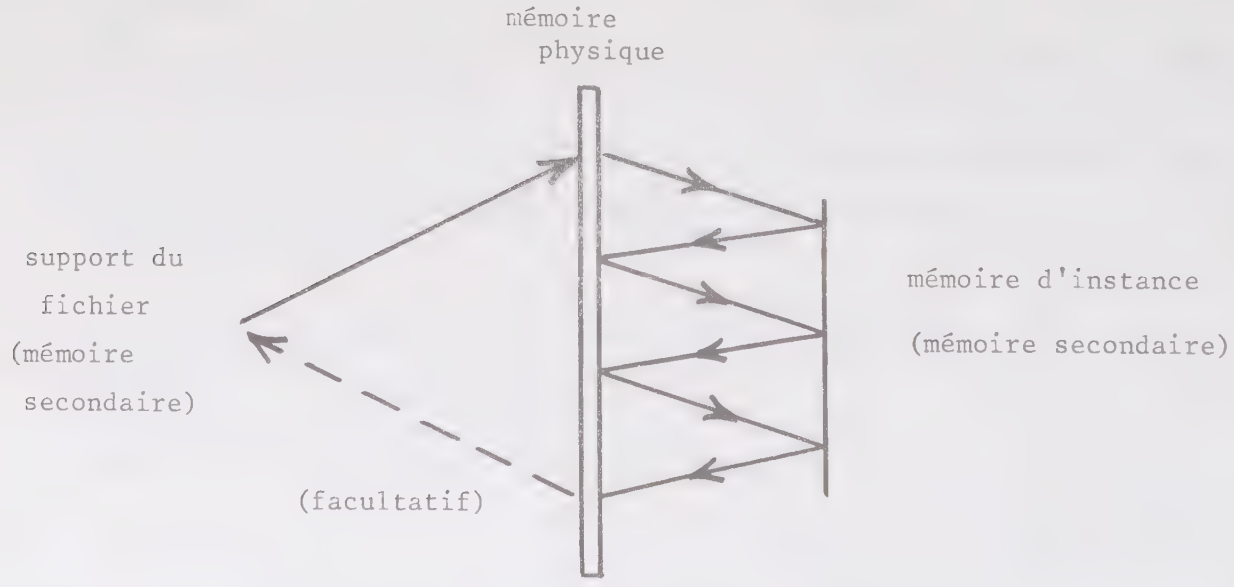


figure 5.4

Les difficultés propres à ce type de solution proviennent essentiellement de la présence de deux mécanismes distincts de transfert entre niveaux de mémoire : un mécanisme d'entrée-sortie (au sens classique) et un mécanisme d'échange ("swapping") pour la gestion de la mémoire virtuelle. Enumérons les principaux inconvénients :

- des copies d'articles de fichiers peuvent figurer simultanément dans la mémoire d'instance et sur le support de fichier : il y a donc perte de place en mémoire secondaire,

- les deux mécanismes de transfert entre mémoire principale et mémoire secondaire ne sont pas indépendants, et des conflits peuvent survenir entre eux (transfert d'une page en mémoire d'instance pendant l'entrée d'un article de fichier dans cette page). Des solutions à ces conflits sont possible, mais aucune n'est simple ni très satisfaisante (cf discussion plus détaillée pour ESOPE dans [B4]).

Les considérations qui précèdent nous ont conduits au choix d'un mécanisme permettant de concilier les avantages des solutions 1 et 2 en essayant d'éliminer leurs défauts respectifs.

a) On réalise l'association

<repère de fichier, numéro d'article>⟶<numéro de page virtuelle>

mais sans imposer un feuillet unique par fichier. Donc aux différentes pages d'une même mémoire virtuelle pourront être associés des articles appartenant à des fichiers différents, et la taille d'un fichier n'est plus limitée par celle de la mémoire virtuelle.

b) L'association précédente n'implique lors de sa réalisation aucun transfert d'information. Un mécanisme unique de transfert est prévu (celui de la gestion de la mémoire virtuelle par défaut de page) et c'est par ce mécanisme qu'un article de fichier est chargé en mémoire physique, et ainsi rendu accessible via la page virtuelle à laquelle il est associé. A un instant donné, un article de fichier a un support physique unique qui peut être soit une page de mémoire physique, soit une page de mémoire secondaire. Le rôle de la mémoire virtuelle est de permettre l'accès à l'information par le mécanisme d'adressage. La mémoire topographique permet d'accélérer l'accès à l'information lorsque le support de l'article du fichier se trouve être la mémoire principale.

On appelle couplage l'opération qui crée le couple (page virtuelle, article de fichier). Un couplage n'entraine aucun transfert d'information, mais simplement la mise à jour d'une table de définition de la mémoire virtuelle.

Pour que l'unité centrale puisse accéder à une information dans un article de fichier, celui-ci doit être couplé à une page virtuelle et chargé

en mémoire principale.

5.23 Le couplage : structures de données et algorithmes

5.231 Structures de données

A chaque mémoire virtuelle (feuillet) doit être associée une table décrivant à tout instant sa composition, c'est-à-dire l'identité des articles de fichiers couplés aux différentes pages virtuelles, ou l'indication de l'absence de couplage. En outre, une protection doit être associée à chaque page couplée.

Ces renseignements sont contenus dans la Table des Pages Virtuelles (TPV), dont il existe un exemplaire par feuillet distinct. A chaque entrée (numéro de page virtuelle) de la TPV, on trouve :

- l'identité de l'article de fichier éventuellement couplé à cette page. Cette identité figure sous la forme d'un doublet (f,a)
  - f est un numéro interne positif identifiant de façon unique tout fichier accessible par couplage ; f = 0 indique que la page n'est pas couplée,
  - a est le numéro d'article du fichier,
- la protection programmée associée à la page (cette information n'a de sens que pour les pages couplées). Les modes de protection possibles sont ceux indiqués au § 5.421.

- l'image du contenu de la mémoire topographique et des registres de protection, recopiée lors de chaque changement de feuillet et constamment tenue à jour dans le but de réduire le temps consacré à ces changements.

Les autres tables sont utilisées pour la localisation physique des divers articles de fichiers.

Elles comprennent une table unique regroupant les renseignements sur tous les fichiers du système (la Table des Fichiers Opérationnels ou TFO), qui sera décrite plus en détail ultérieurement, et une table par fichier indiquant l'implantation physique de chaque article (la Table d'Implantation du Fichier ou TIF). Il suffit ici d'indiquer que l'entrée dans la TFO se fait par le numéro interne f du fichier, et que la TFO contient pour chaque fichier, un pointeur vers le début de la TIF correspondante.

Quant à la TIF, elle comprend essentiellement pour chaque article de fichier une adresse de page physique φ et une adresse de page LOCDISC en mémoire secondaire,

si $\varphi \neq 0$, l'article est en mémoire physique

si $\varphi = 0$, l'article est en mémoire secondaire, à l'emplacement LOCDISC.

De façon plus précise, une copie de l'article existe toujours en mémoire secondaire. Mais si une copie existe en mémoire physique, alors c'est cette copie qui est à jour.

Il est maintenant possible de donner une présentation synthétique (fig. 5.5) du mécanisme d'accès à l'information, en utilisant les structures de données introduites. Ce mécanisme transforme une adresse <feuillet, adresse virtuelle> en une adresse <numéro de fichier, numéro d'article> (le feuillet étant identifié par le contenu actuel des registres topographiques et par l'adresse de la TPV). L'accès effectif est réalisé lorsque l'information se trouve en mémoire physique ; les opérations à effectuer dans le cas contraire seront décrites au chapitre suivant.

Remarquons que la voie normale d'accès à un article couplé est la chaîne (programmée) TPV-TFO-TIF, mais qu'un "court-circuit" est possible en utilisant la mémoire topographique. Remarquons aussi qu'en l'absence de mécanisme câblé de détection de la présence d'une page en mémoire physique, on doit utiliser le déroutement pour violation de protection virtuelle. Le détail de cet algorithme sera donné au § 6.43.

Il reste à examiner la première étape de la correspondance entre information et support, c'est-à-dire le passage

<nom de client, nom de fichier> ⟶ <repère interne de fichier>

Cette correspondance se fait de manière classique à travers une famille de catalogues (un par client). Ces catalogues sont eux-mêmes contenus dans un fichier particulier, accessible à tout usager par couplage. La gestion des catalogues et des fichiers des usagers sort du cadre de cet exposé ; un aspect en a été développé au § 5.12 comme exemple de problème de synchronisation. Il suffit ici d'indiquer que tout fichier accessible par couplage possède un numéro interne, qui lui est attribué à sa création ou lors de son transfert depuis un support externe ; ce numéro figure dans le catalogue du client propriétaire du fichier.

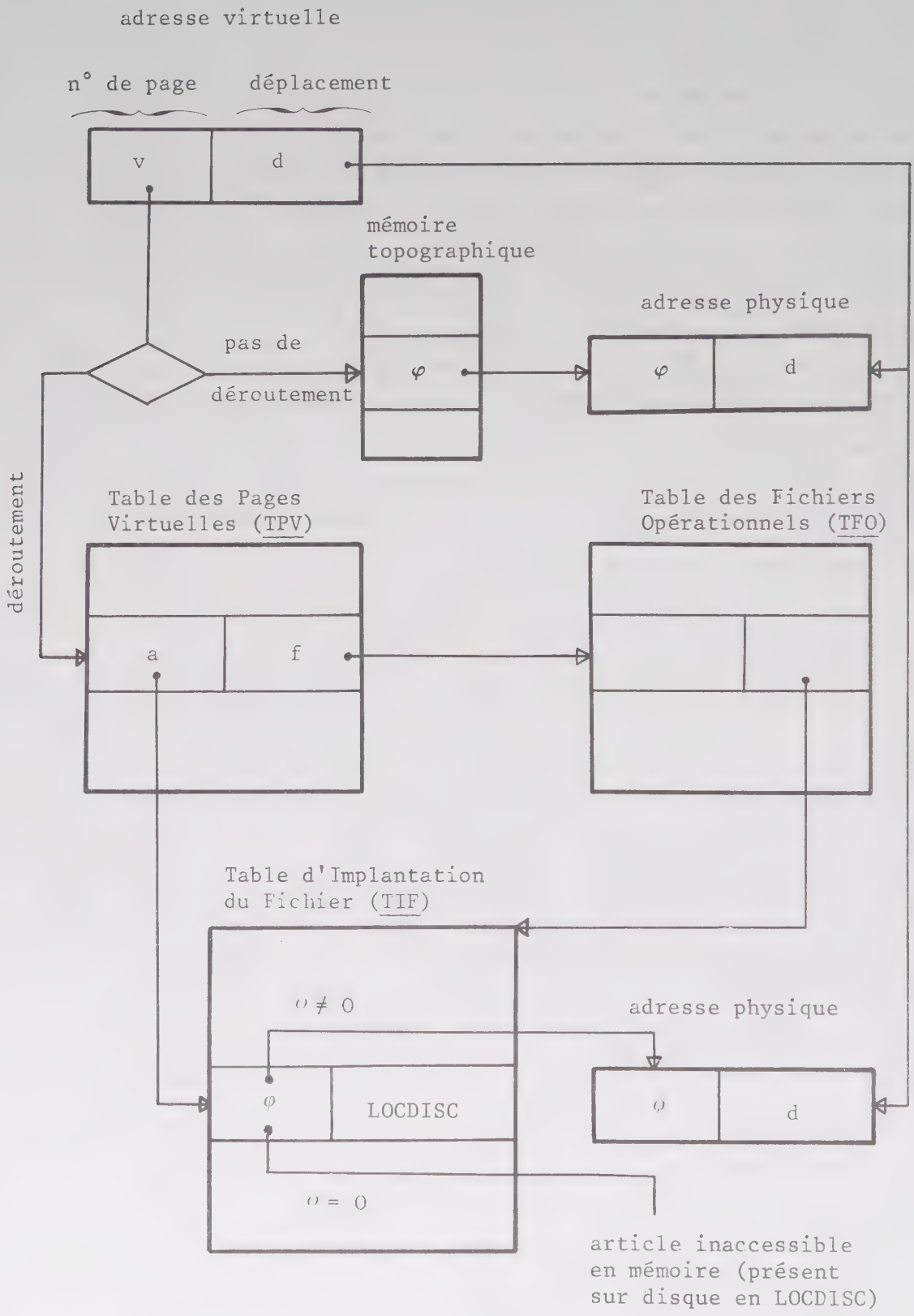


Fig. 5.5 : Mécanisme d'accès à l'information dans le système ESOPE

5.3 Usagers et machines virtuelles

5.31 Choix d'une solution

Les usagers ont été définis en 5.13 comme familles de processus coopérants ; par ailleurs, on a défini en 5.2 le mécanisme permettant à un processeur virtuel (c'est-à-dire à un processus) d'accéder à l'unité d'information, c'est-à-dire à l'article de fichier.

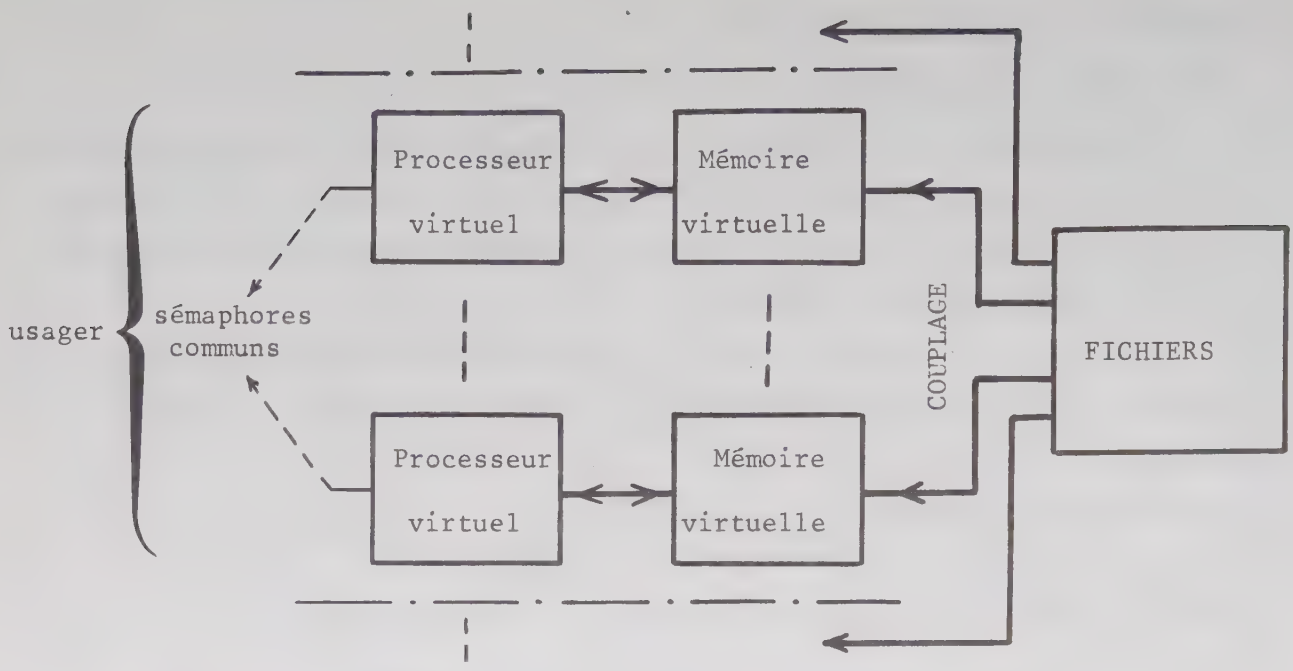
Il reste à établir le lien entre ces deux aspects de la spécification du système en définissant la relation entre processus, usagers, et mémoire virtuelle.

On a en fait le choix entre deux solutions possibles :

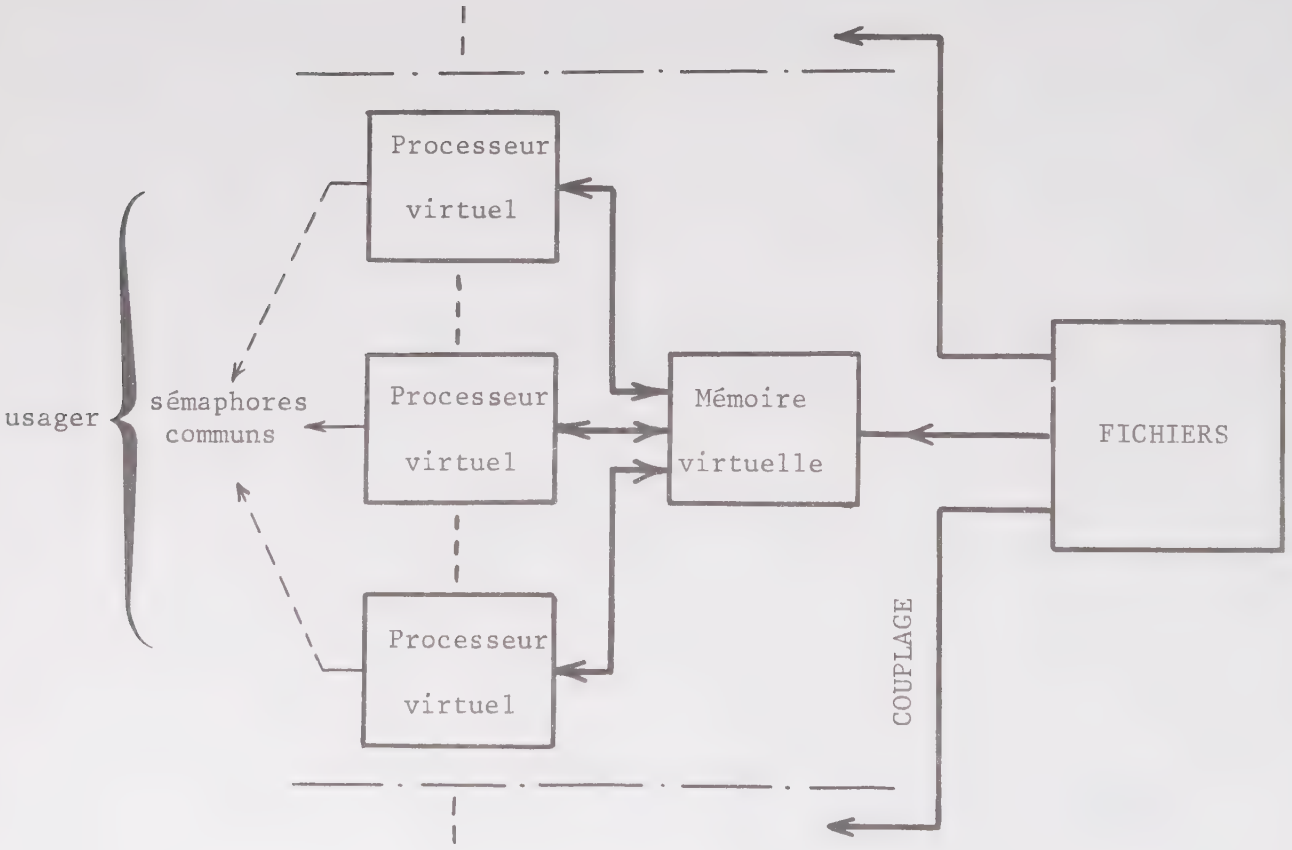
solution 1 : associer une mémoire virtuelle à chaque processus.

solution 2 : associer une mémoire virtuelle à chaque usager.

A chacune de ces solutions correspond un type distinct de machine virtuelle. Les deux solutions sont schématisées sur la figure 5.6



Solution 1 : une mémoire virtuelle par processus



Solution 2 : une mémoire virtuelle par usager

Fig. 5.6 : Relations entre processus et usager

Remarquons qu'au stade actuel de la spécification du système (rien n'étant dit sur l'allocation de ressources) chacune des deux solutions est parfaitement compatible avec les spécifications de 5.1 et 5.2. Nous allons discuter les avantages et inconvénients respectifs des deux solutions et tenter de justifier ainsi le choix effectué.

L'avantage essentiel de la solution 2 est d'apporter une solution immédiate au problème de l'échange d'information entre processus d'un même usager. Cet échange se fait simplement via la mémoire virtuelle commune, alors qu'il nécessite, dans la solution 1, le transit par le système de fichiers en utilisant le couplage. Cette unicité de la mémoire virtuelle a toutefois sa contrepartie : en l'absence d'un dispositif permettant la translation des adresses virtuelles, il n'est pas possible à deux processus appartenant au même usager (dans la solution 2) de partager une procédure commune réentrante. En effet, les adresses de données sont fixées et non translatables, une fois la procédure chargée dans la mémoire virtuelle.

Le partage d'une procédure nécessite soit la recopie pure et simple du texte de la procédure, soit l'artifice peu efficace consistant à n'adresser les données que relativement à un registre utilisé comme base.

Les avantages et inconvénients qui viennent d'être discutés sont d'ordre intrinsèque. Examinons maintenant les contraintes d'implantation.

En ce qui concerne la taille des tables nécessaires, la solution 1 est défavorable, puisqu'il faut prévoir une Table des Pages Virtuelles (TPV) par processus, au lieu d'une par usager dans la solution 2. L'implication la plus délicate de cette situation est la nécessité de trouver, à chaque création de processus, la place nécessaire pour sa TPV. Il n'est pas envisageable, pour des raisons d'économie, de gérer les TPV comme les autres éléments de la Table des Processus (c'est-à-dire avec réservation globale préalable pour un nombre maximal de processus, et attribution par l'algorithme du banquier). L'inconvénient n'existe pas dans la solution 2 où une TPV unique est créée une fois pour toutes à la création de chaque usager et subsiste pendant la vie de l'usager.

Enfin, un argument en faveur d'une mémoire virtuelle par processus est la protection entre processus d'un même usager. La protection des pages virtuelles est en effet liée aux pages et non aux processus qui y font accès (contrairement au mécanisme de protection des pages physiques).

Le tableau ci-après résume les principaux résultats de cette discussion.

	Solution 1 (1 mémoire virtuelle par processus)	Solution 2 (1 mémoire virtuelle par usager)
Avantages	Possibilité d'avoir une procédure réentrante partageable entre processus Protection entre processus d'un même usager.	Partage d'information facile entre processus TPV unique { encombrement limité pas de problèmes à la création de processus
Inconvénients	Partage d'information compliqué entre processus 1 TPV par processus { perte de place problèmes d'allocation de mémoire à la création de processus Temps élevé de commutation entre processus d'un même usager (nécessité de changer les registres topographiques).	Procédure réentrante entre processus d'un même usager : implantation compliquée. Pas de protection a priori entre les processus d'un même usager.

Le choix final d'une mémoire virtuelle par usager a été dicté par des considérations d'efficacité et de simplicité de programmation, qui ont été jugées prépondérantes. La protection entre processus d'un même usager est rendue plus difficile ; toutefois, (cf § 5.4) on a garanti la protection du processus premier, et on a laissé entièrement à la responsabilité de chaque utilisateur la protection de ses autres processus. Un tel choix peut certainement être discuté, et, en fait, ses difficultés peuvent conduire à mettre en cause moins la notion même d'usager que son adéquation au matériel disponible.

5.32 Propriétés des usagers

L'usager est donc à présent défini par les attributs suivants :

- une mémoire virtuelle unique, fonctionnant suivant le mécanisme exposé au § 6.2,

- une famille de processus coopérants, descendants d'un processus unique (le processus premier), et pouvant accéder à des sémaphores locaux à l'usager et à des sémaphores globaux.

Examinons les conséquences de cette structure sur les communications d'information entre usagers et système. On peut distinguer trois types d'informations :

- communes à tous les usagers et au moniteur,
- communes au moniteur et à un usager donné,
- communes à plusieurs usagers.

Le premier cas a été traité en décidant que les deux premières pages de la mémoire physique (pages 0 et 1) font partie des mémoires virtuelles de tous les usagers.

Pour le second cas, on a réservé pour chaque usager une page physique (dite page de garde), associée à la même page virtuelle (page 255) pour tous les usagers.
La page de garde est la première page allouée lors de la création d'un nouvel usager et elle n'est libérée qu'à la disparition de l'usager.

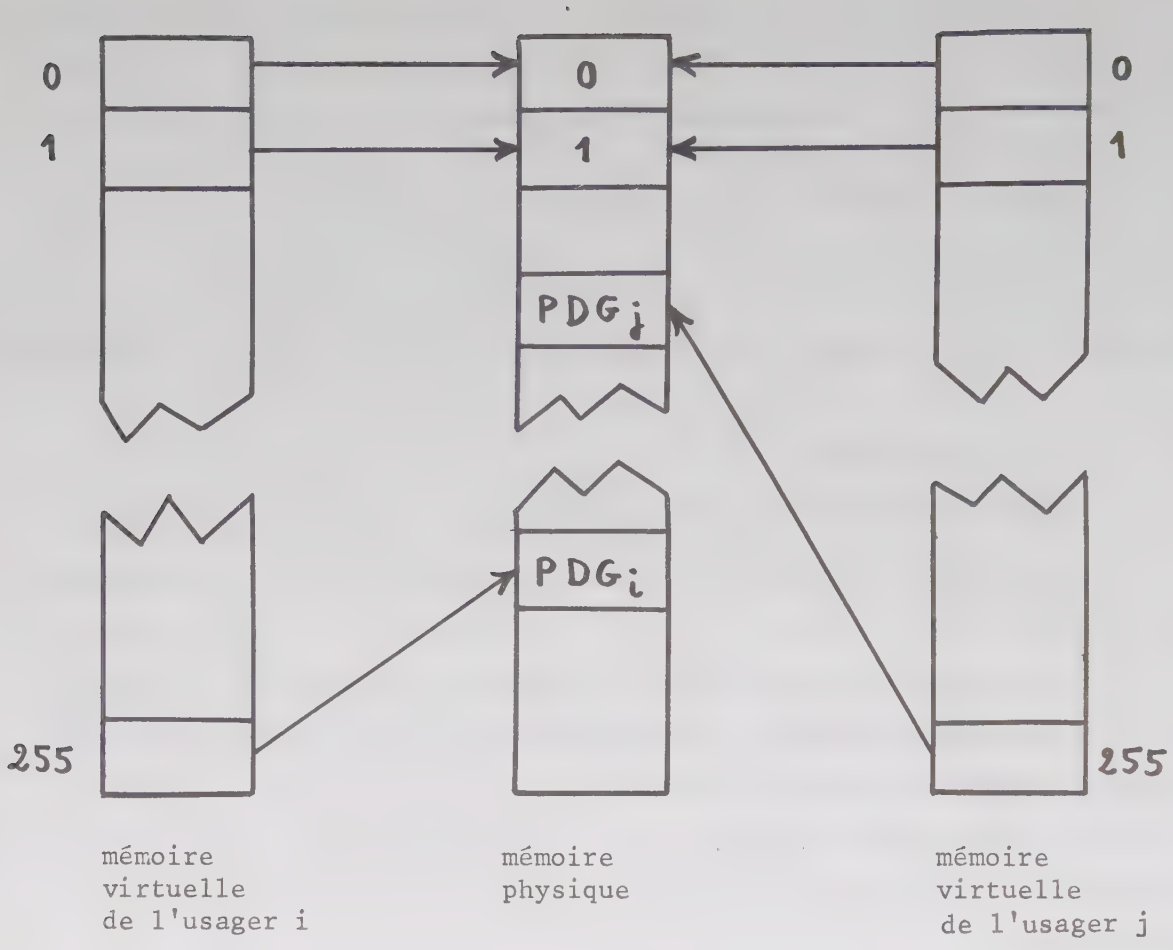


fig. 5.7 : Pages de communication usager-système

La page de garde contient un certain nombre de tables propres à l'usager, en particulier la Table des Pages Virtuelles, et, pour les usagers conversationnels, les tampons d'échange avec les téléimprimeurs.

Le fait de placer la page de garde à la même adresse virtuelle pour tous les usagers permet d'écrire de façon réentrante le programme du processus premier, et, de façon plus générale, ceux des programmes du moniteur qui s'exécutent en mémoire virtuelle.

Les pages 0 à 1 contiennent des informations communes qui doivent être accessibles par leur adresse physique (moniteur) ou virtuelle (usagers). Quant aux informations communes à plusieurs usagers, il s'agit essentiellement des fichiers partageables, et en particulier des programmes réentrants. La figure 5.9 schématise le partage par deux usagers d'un programme commun.

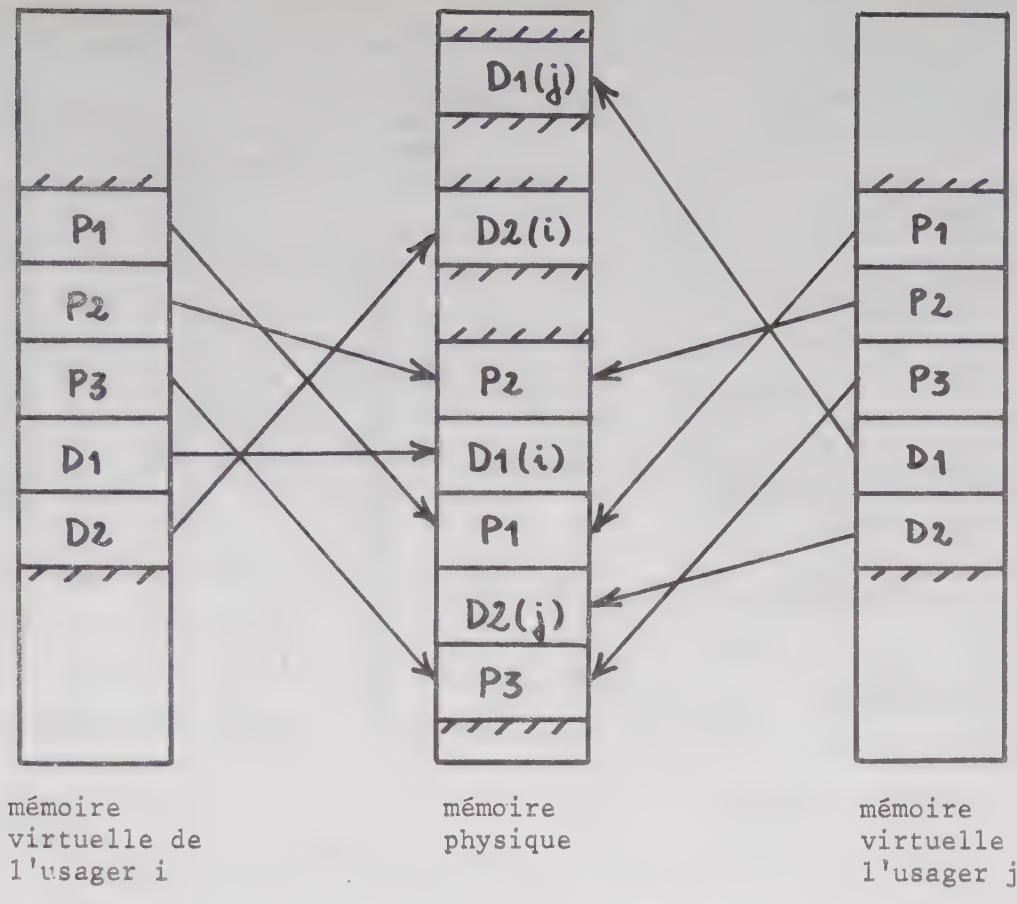


Fig. 5.8 : Partage d'une procédure entre usagers

L'absence d'un mécanisme de translation d'adresse en mémoire virtuelle impose de charger aux mêmes adresses virtuelles chez tous les usagers, un programme partageable et ses données. Le programme n'existe qu'à un exemplaire en mémoire physique.

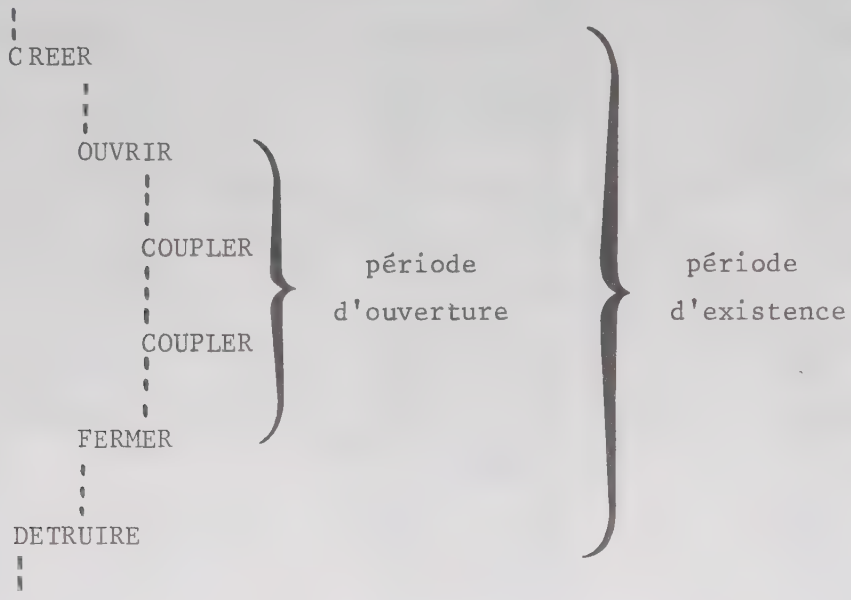
Dans le système ESOPE, tous les programmes exécutés en adresses virtuelles ont été écrits de façon réentrante. Leur partage est donc possible, en respectant la contrainte du chargement à une adresse virtuelle fixe.

5.33 Opérations sur les fichiers

L'organisation du système de fichiers d'ESOPE ne sera pas traitée ici. On se bornera à donner la liste des principales opérations "logiques" (vues par le programmeur) et le principe des opérations d'ouverture, fermeture et couplage, qui intéressent directement la gestion de la mémoire virtuelle, et interviennent dans l'allocation de mémoire physique.

5.331 Opérations logiques sur les fichiers

Le programmeur peut faire sur un fichier un certain nombre d'opérations schématisées ci-après avec leur ordre d'exécution :



Du point de vue du programmeur :

- CREER un fichier, c'est lui donner un nom. A partir de sa création, et jusqu'à sa destruction, un fichier existe dans le système, et son nom permet de lui faire référence.

- OUVRIR un fichier, c'est spécifier son mode d'accès (écriture, lecture...) et autoriser le couplage de ses articles conformément à ce mode d'accès.

- FERMER un fichier supprime la possibilité de couplage à des articles de ce fichier mais aussi l'accès aux articles qui auraient déjà été couplés.

- DETRUIRE un fichier supprime toute possibilité de référence ultérieure au fichier et rend son nom disponible (réutilisable pour créer un autre fichier).

Ces opérations peuvent être assorties de certaines restrictions (en particulier pour les opérations sur les fichiers partagés).

5.332 Ouverture, couplage et fermeture

Pour toutes les opérations sur les fichiers, on a adopté le principe consistant à faire tous les tests de validité de l'opération avant d'exécuter les actions proprement dites. Nous ne détaillerons pas ici ces tests, qui comprennent la vérification des paramètres de l'appel et des droits de l'usager appelant. De même, nous supposerons que les fichiers sont désignés par leur numéro interne F dans le système, laissant de côté le mécanisme du passage du

nom au numéro interne.

```
Ouverture
    procédure ouvrir (F, protection) ;
        begin
        tests de validité ;          / * sortie en cas d'erreur  */
        inscrire protection ;
        NUSL(F) := NUSL(F) + 1 ;
        NUSC(F) := NUSC(F) + 1 ;
        if (fichier commun) & (NUSC(F)= 1) then réservation de mémoire(F);
        end
```

Les compteurs NUSL(F) et NUSC(F) désignent respectivement le nombre total d'usagers et le nombre d'usagers courants pour lesquels F est ouvert. La réservation de mémoire pour F, qui a lieu dans le cas de la première ouverture d'un fichier partagé du système, sera examinée au chapitre suivant.

```
Couplage
    procédure coupler (PV, F, A, N, P) ;
        begin
        / * couple N articles consécutifs du fichier F à partir de
            l'article A, à N pages virtuelles à partir de la page PV,
            avec la protection P
          */
        tests de validité ;
        for K := 1 step 1 until N do
            begin
            FTPV(PV) := F
            ATPV(PV) := A ;
            PETPV(PV := '11'
            PFTPV(PV) := P
            A := A + 1 ; PV := PV + 1 ;
            end ;
        charger PETPV dans les registres de protection de la mémoire
        topographique
        end ;
```

```
Fermeture
     procédure fermer (F) ;
          begin
          tests de validité ;
          NUSC(F):= NUSC(F)- 1 ;
          NUSL(F):= NUSL(F)- 1 ;
          if NUSC (F) = 0 then
               libérer pages physiques du fichier;
          "découpler" les pages couplées du fichier F ;
          end
```

La procédure de libération des pages physiques du fichier lorsqu'il n'est plus ouvert par aucun usager, sera détaillée au chapitre suivant.

5.4 Mécanismes de protection dans le système ESOPE

5.41 Introduction

La nécessité de dispositifs de protection dans un système informatique est l'inévitable contrepartie du partage d'informations et de ressources communes entre de multiples activités concurrentes.
D'une façon plus générale, la fiabilité d'un système est conditionné :

- par le soin et la rigueur apportés à la définition d'une structure logique,
- par la qualité des mécanismes de protection prévus.

Les mécanismes de protection sont apparus dans les calculateurs à partir du moment où plus d'un programme devait utiliser les ressources et accéder à l'information ; initialement, il s'agissait essentiellement de protéger le moniteur contre les erreurs (ou malveillances) des usagers. La situation est devenue plus complexe à mesure que se développait l'emploi partagé, (multiprogrammation et partage de temps) et on peut actuellement distinguer [W1] quatre aspects de la notion de protection :

- Protection du système vis à vis des processus des usagers : aucune possibilité ne doit être laissée matériellement aux usagers de détruire des informations servant à la gestion du système, et plus généralement de perturber ou d'arrêter son fonctionnement,

- Protection mutuelle des usagers : aucune action d'un usager ne doit pouvoir compromettre le déroulement des processus d'un autre usager,

- Protection interne de chaque usager : à partir du moment où chaque usager peut gérer des processus parallèles, se posent pour lui des problèmes analogues à ceux rencontrés par le moniteur vis à vis des différents usagers. Le système doit donc lui fournir les outils nécessaires au traitement de ces problèmes,

- Protection interne du système : une première protection dout être assurée par la rigueur de la conception : toutefois des défaillances sont possibles, soit dans cette conception même (par exemple, la décision d'admettre avec une probabilité faible mais non nulle l'occurence d'un événement catastrophique) soit dans l'implantation (erreur de programmation passée inaperçue) soit enfin dans le fonctionnement du matériel. Dans tous ces cas, on doit pouvoir :

- détecter l'erreur et la localiser avec le plus de précision possible,

- limiter ses conséquences irréversibles.

La réalisation des fonctions de protection dans un système repose de façon générale sur deux types de restrictions :

- restrictions portant sur les actions autorisées,
- restriction portant sur l'accès à l'information.

Des dispositifs câblés permettant de mettre en oeuvre ces deux types de restrictions existent à présent sur la plupart des machines. Avant d'examiner les problèmes de protection propres au système ESOPE, nous décrirons succinctement les dispositifs câblés de protection disponibles sur le calculateur 10070.

5.42 Mécanismes de protection du 10070

5.421 Restrictions des actions

On distingue deux modes de fonctionnement de l'unité centrale : mode maître et mode asservi. Le mode en cours est indiqué par un bit du double-mot d'état programme (PSD). Dans le mode asservi, l'exécution d'un certain nombre d'instructions (dites privilégiées) est interdit, et toute tentative d'exécution se traduit par un déroutement. Les instructions privilégiées sont :

- les instructions d'entrée-sortie,
- les instructions permettant la commande des niveaux d'interruption (armement, déclenchement,...) et l'entrée-sortie directe via les registres,
- les instructions d'échange ou de chargement du PSD,
- les instructions de manipulation de la mémoire topographique et

des registres de protection.

5.421 Restriction des accès à l'information

a) En mémoire physique

A toute page physique est associé un verrou de deux bits ; à tout programme en exécution est associée une clé de deux bits faisant partie du PSD.

La protection de la mémoire physique est assurée de la façon suivante : si la clé d'un programme est égale à :

- 0 , le programme peut écrire dans toute page physique, quelle que soit la valeur du verrou associé à cette page,
- X ≠ 0, le programme ne peut écrire que dans les pages dont le verrou possède la valeur X ou la valeur 0.

La restriction porte donc seulement sur l'écriture. Une tentative d'écriture non autorisée se traduit par un déroutement. La protection par clé-verrou s'applique en mode maître comme en mode asservi.

b) En mémoire virtuelle

A toute page virtuelle est associée une "protection virtuelle" de deux bits. Cette protection est applicable aux programmes fonctionnant en mode "avec topographie" ; elle ne s'applique qu'aux programmes exécutés en mode asservi. (cette particularité rend peu utilisable la configuration "mode maître avec topographie").

Les protections possibles sont les suivantes :

- 00 : tout accès possible
- 01 : lecture et exécution permises
- 10 : lecture seule permise
- 11 : tout accès interdit

c) En mémoire secondaire

Les disques rapides RAD sont organisés en pistes (512 pistes par disque) ; les pistes peuvent être verrouillées manuellement contre l'écriture. Toute tentative de violation de cette protection se traduit par un déroutement.

5.43 Pouvoirs des processus

A tout processus du système est associé un ensemble de pouvoirs spécifiant l'ensemble des actions permises à ce processus. Un principe élémen-

taire de sécurité consiste à n'attribuer à tout instant à un processus que les pouvoirs les plus restreints compatibles avec la bonne exécution de sa tâche. Toute action nécessitant une extension temporaire des pouvoirs d'un processus doit se faire sous le contrôle du système.

Les dispositifs câblés disponibles pour représenter les pouvoirs d'un processus sont constitués par plusieurs éléments de son double-mot d'état de programme :

- les masques d'inhibition des déroutements (arithmétique, décimal,...)
- les masques d'inhibition des interruptions (par groupe de niveaux)
- la clé d'écriture en mémoire physique
- les indicateurs de mode : maître-asservi et physique-virtuel.

Ces possibilités sont insuffisantes pour tous les besoins recensés au § 6.41. On les a donc complétées, dans le système ESOPE, par un système plus général de protection programmée dont le schéma d'ensemble est le suivant :

- les processus sont divisés en quatre classes ; à chaque classe est attaché un ensemble d'actions permises,
- à tout processus est attaché un ensemble de pouvoirs dits intrinsèques, inclus au sens large dans l'ensemble des pouvoirs associés à la classe à laquelle il appartient,
- un processus peut disposer temporairement de pouvoirs plus étendus que ses pouvoirs intrinsèques, mais uniquement au cours de l'exécution de séquences de programmes du système, protégées contre tout incident de nature technologique.

Ces opérations sont programmées sous la forme d'"appels au superviseur" (instructions CAL de la 10070) qui permettent le branchement à une adresse prédéterminée avec modification éventuelle du reste du PSD. Les premières instructions exécutées après l'appel ont pour but de vérifier si le processus est autorisé à exécuter l'action; on vérifie pour cela sa classe, puis éventuellement ses pouvoirs intrinsèques.

Les quatre classes de processus sont :

0 - les processus résidents du système (processus de l'allocateur et processus d'entrée-sortie)
1 - les processus premiers des usagers
2 - les processus des traducteurs du système
3 - les autres processus des usagers.

Examinons les pouvoirs attachés à chacune de ces classes vis à vis des différentes actions possibles :

- Primitives de synchronisation : P, V, D

Suivant la classe du processus, le sémaphore pris en compte est déterminé

- par son indice dans la table générale des sémaphores, pour les processus des classes 0 et 1,
- par son nom interne à l'usager appelant la primitive, pour les processus des classes 2 et 3.

On garantit ainsi qu'un processus ordinaire d'un usager ne peut agir que sur les sémaphores locaux à cet usager. En revanche, le processus premier ne peut pas accéder par nom à ces sémaphores locaux, mais il peut agir sur les sémaphores du système.

- Autres primitives portant sur les processus et sémaphores des usagers

Toutes ces actions sont interdites aux processus résidents du moniteur. Les primitives permettant de consulter l'état, de créer ou détruire des processus et sémaphores d'un usager sont autorisées à tous les processus de cet usager. Les primitives permettant de modifier le vecteur d'état des processus sont permises dans tous les cas sauf dans le sens processus quelconque → processus premier.

Sont enfin réservées au seul processus premier les actions suivantes :

- détruire tous les processus et sémaphores d'un usager
- interrompre tous les processus d'un usager et les placer dans la file d'un sémaphore global (un par usager) réservé à cet usage.

Cette dernière action répond à un souci de sécurité : quelles que soient les erreurs commises par les processus d'un usager (boucle infinie, "étreinte fatale",...) il doit être possible, au moins pour les usagers conversationnels, de revenir à un état de référence stable, permettant la détection et la correction éventuelle des erreurs. Cela est réalisé dans ESOPE par le passage du contrôle au processus premier et l'interruption, au sens qui vient d'être précisé,de tous les autres processus de l'usager. On se reportera à [F1] pour les détails techniques de ce mécanisme.

- Primitives portant sur les fichiers

Ces primitives ne sont pas autorisées aux processus résidents du moniteur. Pour les autres processus, les restrictions sont schématisées sur

la figure 5.9.

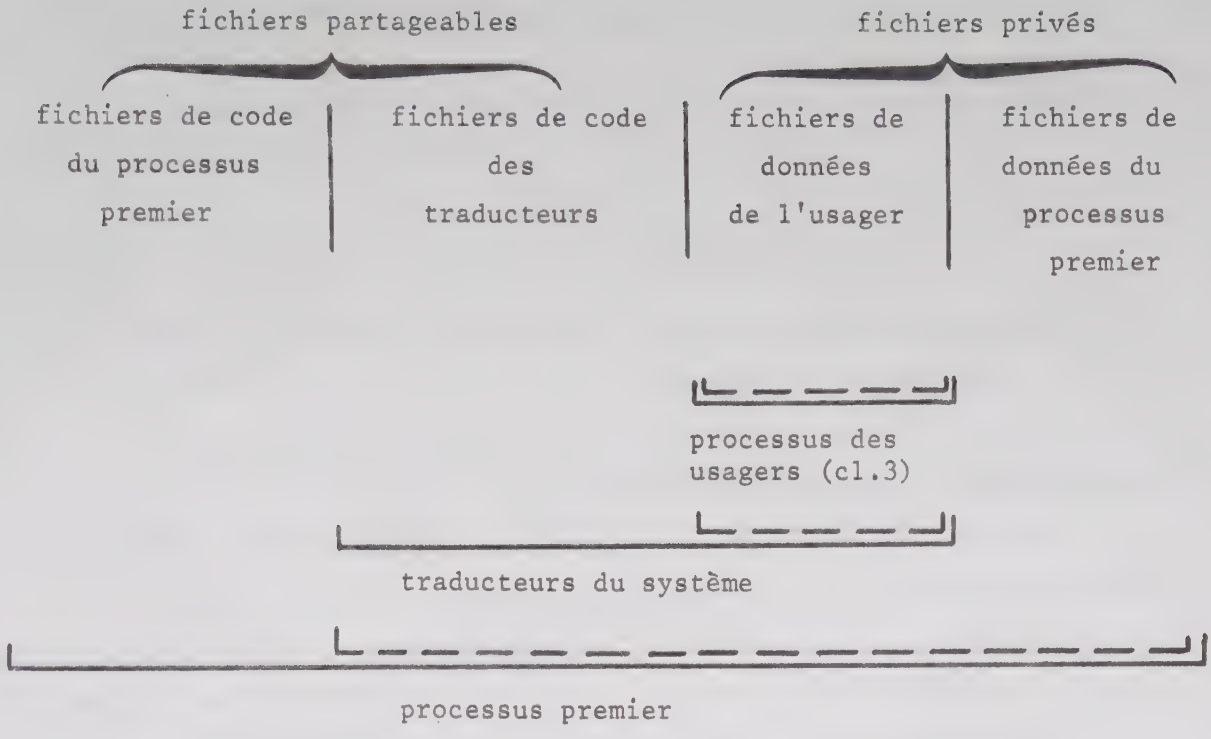


Fig. 5.9 : Restriction des actions sur les fichiers

Les restrictions portant sur les opérations d'ouverture, fermeture, couplage, sont indiquées en traits pleins, celles (plus sévères) portant sur la création, la destruction, la modification des attributs, en traits pointillés.

5.44 Protection de l'information

5.441 Protection interne d'un usager

A l'intérieur de la mémoire virtuelle d'un usager les informations accessibles aux processus de l'usager sont :

- les zones de communication (pages 0, 1, page de garde). Leur protection est assurée par le mécanisme clé-verrou (voir plus loin)

- les pages virtuelles couplées à des articles de fichiers.

Comme on ne dispose pas pour la mémoire virtuelle de dispositif analogue aux clés et verrous de la mémoire physique, la protection mutuelle des

zones de données des divers processus d'un usager doit être directement programmée par cet usager, en utilisant les primitives d'exclusion mutuelle et le couplage.★ Toutefois, la protection des données privées du processus premier est toujours assurée, grâce aux dispositions suivantes :

- le processus premier ne peut pas être actif en même temps qu'un autre processus de l'usager,

- le fichier de données privées du processus premier ne peut être ouvert et couplé que dans deux circonstances:
  - pendant l'activité du processus premier
  - au cours de l'exécution par un processus quelconque d'une primitive protégée par un CAL (le processus exécute alors un programme du système).

5.442 Protection de la mémoire physique

Les deux tableaux ci-après résument l'utilisation des clés et verrous de la mémoire physique:

	Verrous		Clés
00	non utilisé	00	attribuée temporairement (uniquement à l'intérieur d'un CAL)
01	données des usagers	01	Processus des usagers (y compris processus premier)
10	Données résidentes pages 0 et 1 pages de garde des usagers	10	Processus du moniteur
11	pages de programme (résident ou non) données utilisées uniquement à l'intérieur des CAL	11	non utilisé

Utilisation du mécanisme de protection contre l'écriture en mémoire physique

★ Cette restriction constitue l'inconvénient principal de l'implantation de la notion d'usager dans ESOPE

Les pages allouées temporairement conservent la valeur de leur verrou jusqu'à leur réallocation.

On voit que tous les types de protection énumérés au § 5.41 (sauf la protection interne programmée par l'usager, cas déjà mentionné) sont ainsi résolus. En effet :

- aucun processus (du moniteur ou d'un usager) ne peut écrire dans une page contenant du code exécutable,

- aucun processus d'un usager ne peut écrire dans les pages de communication de sa mémoire virtuelle (0, 1, page de garde),

- aucun processus du moniteur ne peut écrire dans les pages de données d'un usager.

(la protection entre usagers et la protection de la partie résidente contre les usagers résulte de la réalisation des mémoires virtuelles par la correspondance topographique).

Dans tous les cas où les protections ci-dessus doivent être temporairement violées, elles le sont par l'intermédiaire de primitives réalisées par des CAL, dont l'utilisation elle-même est contrôlée : c'est le cas (par exemple) de l'attribution temporaire à un processus premier de la clé 10, pour écrire dans les zones de communication de son feuillet.

Chapitre 6

CONCEPTION ET REALISATION DE L'ALLOCATEUR

Dans ce chapitre, nous examinons comment les principes de conception de l'allocateur d'ESOPE ont été établis en fonction des choix fondamentaux exposés aux deux chapitres précédents. Nous montrons ensuite comment les considérations développées dans la première partie de ce travail ont déterminé le schéma central de l'allocation de ressources, matérialisé par le cycle des usagers. Enfin, nous décrivons plus en détail les principaux mécanismes mis en oeuvre. Nous renvoyons à la bibliographie pour les développements les plus techniques.

6.1 Principes de l'allocation de ressources.

A l'issue du chapitre précédent, nous avons défini la notion d'usager et la structure de la machine virtuelle attachée à un usager. La fonction de l'allocateur de ressources est de matérialiser ces machines virtuelles en leur associant au cours du temps les ressources physiques nécessaires.

Les ressources physiques partageables comprennent la mémoire principale, la mémoire secondaire et l'unité centrale. (Nous avons vu que les périphériques n'entraient pas dans la catégorie des ressources partageables). Par ailleurs, la notion de partage s'applique aussi à des ressources immatérielles, comme certains fichiers ou certaines tables du système.

D'après la structure de la machine virtuelle, l'allocataire de mémoire est l'usager. Cette structure n'impose pas par elle-même que l'usager soit également allocataire du processeur, et on pourrait concevoir un schéma où le processeur serait alloué individuellement aux processus, indépendamment de la notion d'usager. Ce schéma a été écarté pour une double raison :

homogénéité : il est plus simple conceptuellement d'avoir une entité unique comme allocataire des ressources,

efficacité : la commutation de processeur entre processus est beaucoup plus rapide que la commutation de mémoire entre usagers.

Par ailleurs, on facilite la communication d'information entre processus d'un

même usager via leur mémoire virtuelle commune. On peut donc énoncer le premier principe :

1. L'allocation de mémoire centrale et de processeur se fait aux usagers (et non individuellement aux processus).

La conséquence de la structure de mémoire secondaire définie au § 5.2 (pas de zone privilégiée pour la mémoire d'instance, mais sauvegarde de l'information dans le fichier d'origine), se traduit, en termes d'allocation de ressources, par le second principe :

2. La mémoire secondaire est partagée entre les fichiers (fichiers du système et fichiers des usagers).

Pour pouvoir s'exécuter de manière efficace, tout processus a besoin d'une taille de mémoire supérieure à un certain seuil. Il n'est donc pas efficace d'allouer un processeur à un processus s'il n'a pas une probabilité élevée de disposer de cette taille minimale de mémoire. Un second argument dans le même sens est fourni par la disparité, déjà mentionnée, des temps de commutation respectifs pour la mémoire et l'unité centrale. D'où le troisième principe :

3. La mémoire centrale est allouée à un usager avant le processeur.

Remarquons que cette politique élimine le cas de blocage résultant de l'attribution de deux classes distinctes de ressources (mémoire et processeur) à une même famille d'allocataires, les usagers.

Une seconde conséquence de l'existence d'un parachore pour les programmes est le risque de congestion du canal mémoire centrale-mémoire secondaire si un nombre trop élevé de programmes d'usagers coexistent dans la mémoire centrale.

On adoptera donc une politique du type de celles décrites au chapitre 3 :

C'est le quatrième principe :

4. Le nombre d'usagers admis à se partager la mémoire sera limité.

Les algorithmes utilisés pour mettre en oeuvre ce principe de sélection seront examinés dans ce chapitre.

Enfin, des considérations élémentaires de sécurité conduisent à formuler le cinquième principe.

5. Une ressource ne peut être allouée à un usager pour une durée indéfinie.

Toute allocation est faite pour une durée maximale spécifiée à l'avance.

6.2 Le cycle des usagers

6.21 Etats des usagers

Rappelons qu'un processus, défini à l'aide de la notion de machine virtuelle, peut se trouver dans l'un ou l'autre de deux états "intrinsèques".

processus (état virtuel)
- actif : le processus s'exécute sur son processeur virtuel
- bloqué : le processus attend un signal d'activation d'un autre processus.

Tout processus du système ESOPE est inclus dans un usager auquel sont allouées les ressources physiques. Un processus intrinsèquement actif peut dont, suivant les ressources allouées à l'usager dont il faut partie, se trouver dans trois états possibles, compte tenu de l'ordre d'allocation de ces ressources.

processus actif état réel
- bloqué "technologiquement" pour manque de mémoire physique
- prêt, c'est-à-dire disposant de mémoire mais non d'un processeur réel
- élu, c'est-à-dire en cours d'exécution sur un processeur réel.

Réciproquement, l'état d'un usager est défini par l'état intrinsèque de ses processus et par l'ensemble des ressources dont il dispose. Les états suivants sont donc possibles pour un usager :

- oisif : tous ses processus sont bloqués intrinsèquement . Un usager oisif n'est pas demandeur de ressources,
- prêt : au moins un processus est intrinsèquement actif (il y a donc demande de ressources) mais aucune ressource n'est allouée,
- courant : il y a demande de ressources, et il y a eu allocation. Suivant la nature des ressources demandées et allouées, 3 cas sont possibles, correspondant à 3 sous-classes d'usagers courants :
  - suspendu : usager courant en attente de mémoire,
  - éligible : usager courant ayant reçu la mémoire demandée, mais en attente de processeur,
  - élu : usager possédant toutes les ressources demandées.

Le tableau ci-après résume ces définitions :

		Etat des processus de l'usager	Ressources allouées	
			Mémoire	Processeur
	OISIF	tous bloqués	non	non
	PRET	au moins un actif	non	non
COURANT	SUSPENDU	"	(*)	non
COURANT	ELIGIBLE	"	(**)	non
COURANT	ELU	"	(**)	oui

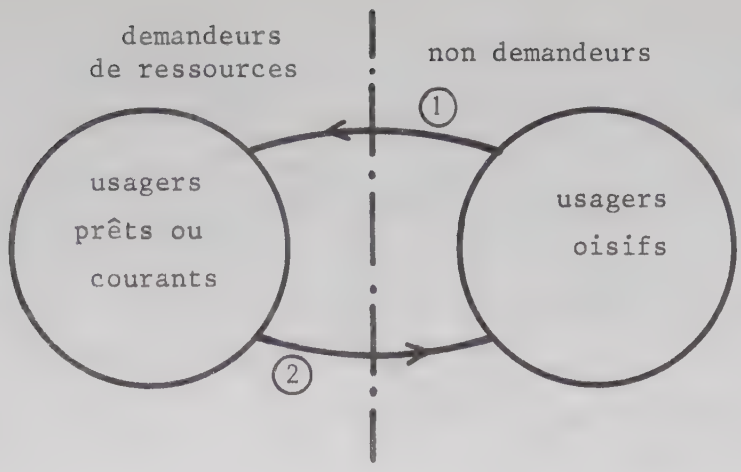
(*) : usager en attente de pages de mémoire

(**) : mémoire allouée, aucune demande de page insatisfaite.

Les transitions entre ces états peuvent être provoquées par deux classes d'événements :

- intrinsèques : activation ou blocage de processus
- technologiques : modification de l'état d'allocation des ressources.

Remarquons que les seules transitions provoquées par des événements intrinsèques sont celles qui partent de l'état OISIF ou y aboutissent.
Toutes les autres mettent en jeu l'allocation des ressources.



(1) déblocage du premier processus de l'usager (les autres sont bloqués)

(2) blocage du dernier processus actif de l'usager.

Fig. 6.1 : Transitions causées par des événements intrinsèques

Les transitions prêt-courant sont régies par le mécanisme qui détermine dans la classe des usagers demandeurs de ressources ceux qui seront servis (mise en oeuvre du 4ème principe).

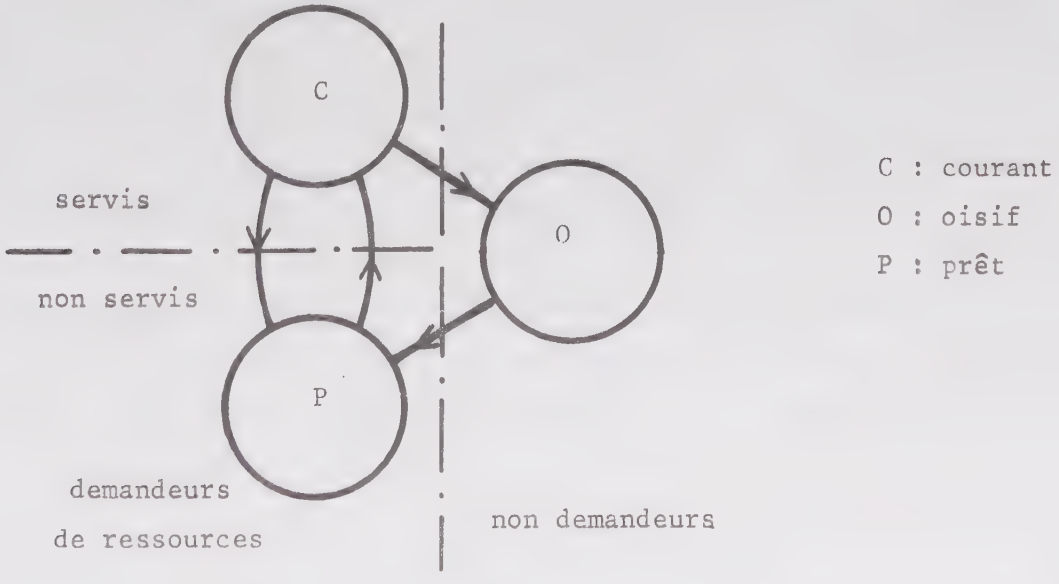


Fig. 6.2.

Sélection des usagers admis au partage des ressources

Enfin, l'allocation effective des ressources gouverne les transitions entre les états des usagers courants :

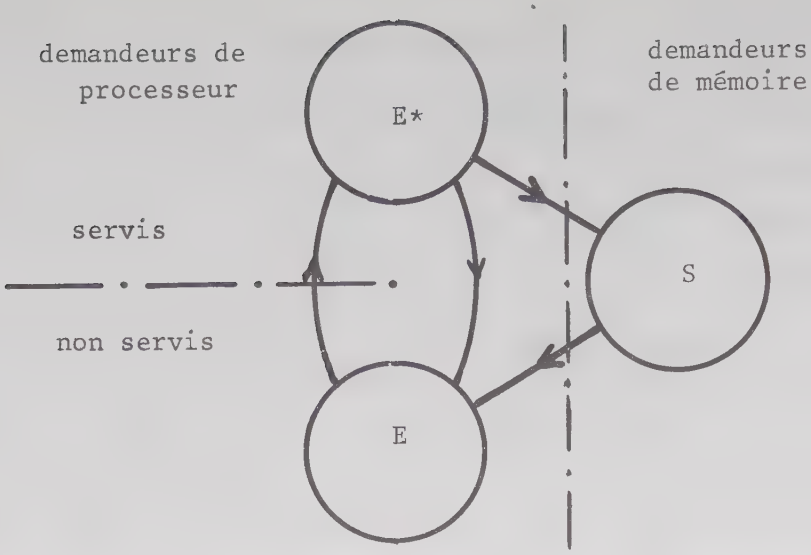


Fig. 6.3. : Cycle des usagers courants

Les transitions permises résultent de l'ordre choisi pour l'allocation et la libération des ressources mémoire et processeur.

Enfin, en faisant la synthèse des deux diagrammes ci-dessus, on obtient le schéma général des transitions entre états d'usagers :

E* : élu
E : éliligible
S : suspendu
O : oisif
P : prêt

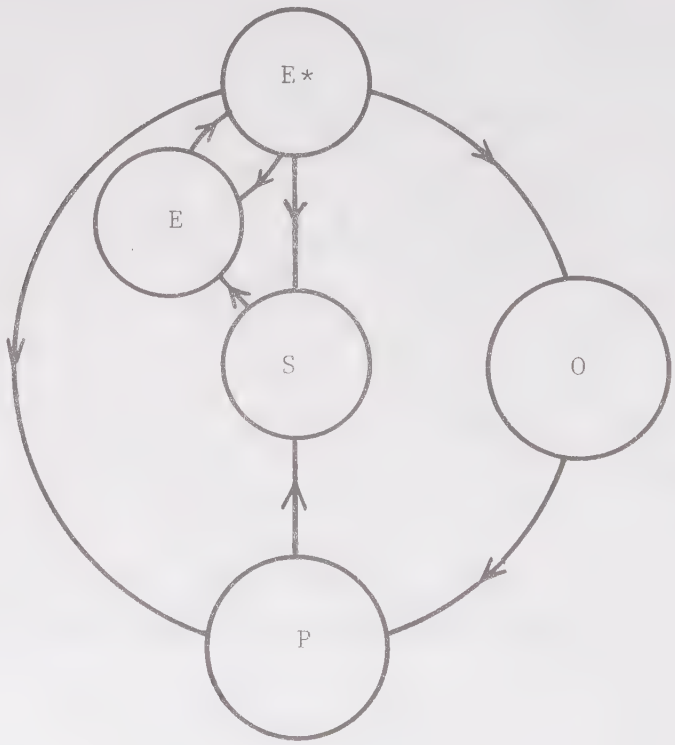


Fig. 6.4. : Le cycle des usagers dans le système ESOPE

6.22 Transitions entre états

On résume ici les différents cas de transition :

O ⟶ P : un usager oisif devient prêt dès qu'un de ses processus est débloqué intrinsèquement

P ⟶ S, E ⟶ P : ces deux transitions dépendent de l'algorithme de limitation du nombre des usagers courants

S ⟶ E : un usager suspendu devient éligible lorsque la mémoire qu'il demande lui est allouée (il n'y a plus de demande de mémoire insatisfaite pour cet usager)

E ⟶ S : aucun processus de l'usager élu ne peut continuer à s'exécuter, (tous ses processus actifs sont bloqués technologiquement)

E ⟶ E*, E* ⟶ E : ces deux transitions dépendent du mécanisme d'allocation d'unité centrale aux usagers

E* ⟶ O : l'usager élu cesse d'être demandeur de ressources (tous ses processus sont bloqués intrinsèquement).

6.23 Etats et files d'usagers

On s'intéresse à présent à la façon de structurer l'ensemble des usagers qui se trouvent dans un même état. Cette structure dépend de la manière dont les divers usagers rejoignent et quittent un état donné (événements que nous appellerons "entrée" et "sortie" dans ce paragraphe).

- usagers oisifs : les entrées et sorties des différents usagers sont provoquées par des événements intrinsèques, (internes à chaque usager), donc indépendants. Aucune structure ne s'impose a priori pour l'ensemble des usagers oisifs.

- usagers prêts : il y a compétition entre ces usagers pour l'entrée dans l'état courant. On introduira donc une file d'attente, la file des usagers prêts, ou file P. La gestion de cette file sera explicitée plus loin.

- usagers courants : les usagers suspendus attendent des pages de mémoire, les usagers éligibles sont en compétition pour l'usage du processeur.
  On pourrait donc construire deux files distinctes.

En fait, grâce à la sélection des usagers courants, les usagers suspendus ne sont pas dans une situation de concurrence, et on montrera d'ailleurs (7.42) que l'algorithme de traitement des requêtes de pages traite les demandes indépendamment de leur ordre d'arrivée. Ces considérations justifient la structure adoptée pour les usagers courants : une file unique (appelée la file C) dont un sous-ensemble (les usagers éligibles) est pris en considération pour l'allocation de processeur.

6.3 Allocation d'unité centrale

L'unité centrale est allouée aux usagers éligibles, qui constituent un sous-ensemble de la file des usagers courants. Dans cette file est choisi l'usager élu. La méthode de choix est très simple : la file étant refermée sur elle-même, l'unité centrale est successivement allouée aux usagers courants dans l'ordre de leur arrivée dans la file C, en sautant les usagers suspendus. L'entrée dans la file C se fait à la suite de l'usager le plus ancien. La sortie de la file C ne peut se faire que dans l'état élu.

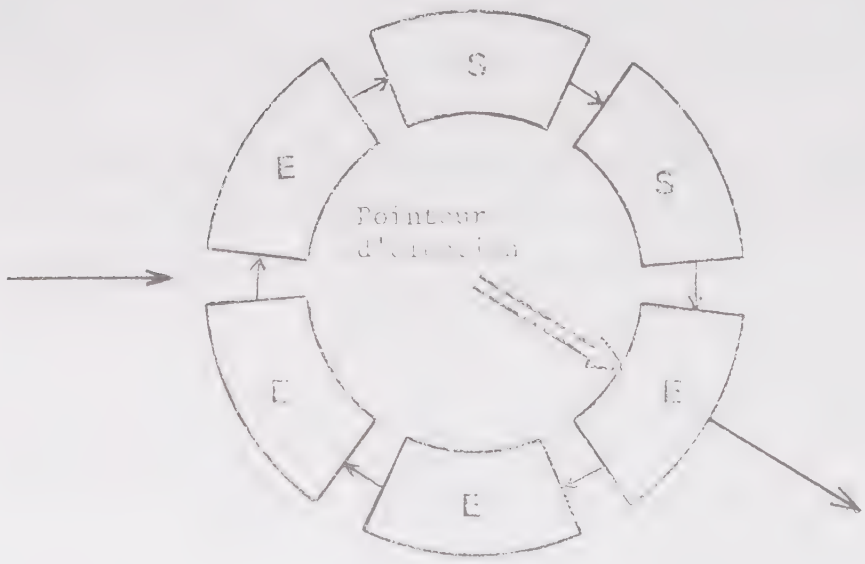


Fig. 6.5 - Structure de la file des usagers courants

En vertu du cinquième principe, l'unité centrale n'est allouée à un usager que pour une durée limitée, ou plus égale à un quantum de temps q.
Le rôle de cette limitation est double :

- assurer la sécurité, en évitant qu'une boucle infinie dans un processus d'un usager n'interrompe le service des autres usagers.

- garantir un temps de réponse minimal à chaque usager.

Il est en fait souhaitable qu'un usager soit interrompu pour fin de quantum le moins souvent possible, puisque le temps de commutation s'inscrit en perte nette. Dans un système essentiellement destiné à un emploi conversationnel, la durée moyenne t d'une interaction et le nombre moyen N d'usagers courants doivent être tels que Nt soit de l'ordre de grandeur d'un temps de prise en compte acceptable, ce qui conduit à l'idée d'un quantum idéal variable avec le nombre d'usagers courants. Sans grande perte d'efficacité, il est plus simple de prendre un quantum fixe, dont l'expérience fixe la valeur dans la plage de 20 à 50ms, dans le cas du système ESOPE.

L'usager élu peut posséder plusieurs processus prêts : ceux-ci se trouvent dans une file d'attente et sont servis selon leur priorité. Un processus qui se bloque perd le processeur au profit du processus suivant ; le blocage du dernier processus prêt, ou la fin du quantum, entraînent le changement d'usager élu. Le mécanisme de ce changement sera examiné plus en détail au chapitre suivant.

6.4 Allocation de mémoire

L'étude de l'allocation de mémoire dans ESOPE peut être divisée en trois parties :

- algorithme de sélection des usagers courants parmi les usagers prêts
- algorithme d'allocation effective des pages de mémoire physique
- organisation des transferts d'information entre mémoire principale et mémoire secondaire.

6.41 Algorithme de sélection des usagers courants

La limitation du nombre des usagers courants a pour but d'assurer à chaque programme en mémoire un nombre de pages suffisant pour pouvoir s'exécuter efficacement ; autrement dit, il s'agit de garantir à chaque programme d'un usager courant une taille de mémoire de l'ordre de grandeur de son parachore, et d'empêcher le "vol" de pages entre usagers.

Le second objectif est atteint en admettant que toute page allouée à un usager courant lui reste allouée tant que l'usager reste courant (ou jusqu'à ce qu'il la libère explicitement). Quant au premier objectif, il est plus difficile à atteindre puisque le parachore d'un programme n'est pas connu à l'avance ; on a choisi d'estimer a priori la taille maximale nécessaire à chaque usager et de réajuster cette évaluation compte tenu du comportement du programme.

La notion de réservation de mémoire a été introduite pour régler les transitions prêt-courant "Réserver" N pages à un usager, c'est lui garantir qu'il pourra occuper pendant son séjour dans l'état courant un nombre de pages physiques au plus égal à N. L'identité des pages n'est pas spécifiée au moment de la réservation. Une ressource partageable ne pouvant être allouée indéfiniment à un usager (5ème principe), toute réservation s'accompagne d'une limitation du temps pendant lequel les pages pourront être allouées. Le temps maximal spécifié, dit temps de résidence, joue à l'égard de la ressource "mémoire physique" le même rôle de garde-fou que le quantum vis à vis de la ressource "processeur". Le couple (N, T) définit une catégorie d'usagers. Les usagers pourront évoluer d'une catégorie à l'autre suivant leur demande effective de mémoire, mesurée au cours de leur passage dans l'état courant. Ce mécanisme réalise une gestion plus souple qu'une limitation statique (par exemple) du degré de multiprogrammation, c'est-à-dire du nombre d'usagers courants.

La réservation se fait au moment de l'admission d'un usager dans la file des courants, ce qui peut arriver dans les circonstances suivantes :

a) libération de mémoire physique (consécutive au départ d'un usager de la file C ou à la fermeture d'un fichier partagé)

b) entrée d'un usager dans la file P alors que celle-ci est vide.

La condition d'admission dans la file C est :

$$N_{max} \leq N_{disp}$$

où
- N_{max} désigne la taille maximale spécifiée par la catégorie de l'usager.
- N_{disp} désigne le nombre de pages physiques disponibles, c'est-à-dire non réservées.

Si cette condition est satisfaite pour le premier usager de la file P, celui-ci entre dans la file C, où il restera au plus jusqu'à l'écoulement de son temps de résidence. On fait alors

$$N_{disp} := N_{disp} - N_{max}$$

et le test d'admission est répété pour l'usager suivant, jusqu'à échec.
Si le test échoue pour le premier usager de la file P, aucun usager n'est admis dans la file C (cela pour éviter les cas d'attente indéfinie pour des usagers ayant ayant un grand N_{max}).

L'entrée d'un usager dans la file C s'accompagne d'une allocation initiale de pages, et d'une demande de chargement, dans des conditions qui seront examinées au § 6.4 . L'usager passe dans l'état "suspendu" en attendant l'exécution du chargement.

Cas des fichiers partagés

Un fichier partagé est caractérisé par le nombre N_f de ses pages.

La réservation de mémoire pour un fichier partagé peut être faite dans deux cas :

a) à l'entrée dans la file C d'un usager ayant ouvert le fichier (si une réservation n'a pas déjà été faite pour ce fichier).
Le test d'admission de cet usager devient dans ce cas :

$$N_{max} + N_f \leq N_{disp}$$

Si le test réussit, on fait

$$N_{disp} := N_{disp} - (N_{max} + N_f)$$

et la réservation de N_f pages restera valable pour le fichier tant que celui-ci restera ouvert pour au moins un usager courant (il n'y a donc pas de temps de résidence préassigné pour les fichiers partagés).

b) lors de l'ouverture par l'usager élu d'un fichier partageable qui n'est pas déjà ouvert pour un usager courant.
On vérifie alors si

$$N_f \leq N_{disp}$$

Si oui, N_f pages sont réservées pour le fichier, et on fait

$$N_{disp} := N_{disp} - N_f.$$

Si non, l'usager demandeur est renvoyé dans la file P, et on sera ramené au cas a) lors de sa réadmission dans la file C (sauf toutefois si la réservation de pages pour le fichier partagé a pu

être faite entre temps par un autre usager).

Enfin, la transition courant-prêt est déclenchée pour tout usager qui tente de dépasser l'une des deux bornes supérieures fixées par sa catégorie.

Les changements de catégorie sont provoqués comme suit :

- en cas de dépassement de Nmax, l'usager est transféré, à son retour à l'état prêt, dans la catégorie comportant un Nmax immédiatement supérieur.
- de même, en cas de surestimation de Nmax (c'est-à-dire si le nombre de pages effectivement utilisées est inférieur à celui correspondant à la catégorie inférieure, l'usager est transféré dans cette catégorie.

Critique de l'algorithme de réservation

L'algorithme qui vient d'être exposé est simple et répond au but proposé mais il n'est pas à l'abri de toute critique. En particulier, la mémoire physique n'est apparemment pas utilisée de façon optimale. En effet :

- lors de la première réservation pour un fichier partageable, la réservation est faite pour la totalité du fichier.
- un usager peut être éliminé de la file C à la suite d'un dépassement de son Nmax alors que des pages physiques sont encore disponibles.

En ce qui concerne les fichiers partageables, on peut faire les remarques suivantes :

- l'inconvénient de la réservation globale est réduit si on limite (par découpage) la taille des fichiers partagés.
- On peut améliorer l'algorithme de réservation en prenant pour N_f un nombre de pages inférieur à la taille du fichier partagé, mais en autorisant le dépassement ; les pages excédentaires sont prélevées sur le "crédit de pages" de l'usager demandeur, c'est-à-dire la différence entre son N_{max} et le nombre de pages effectivement réservées. En ce qui concerne la sévérité du test d'élimination, on pourrait songer à allouer des pages à un usager tant que $N_{disp}>0$. Le risque est alors qu'un comportement aberrant d'un seul usager (erreur ou malveillance) ne dégrade le temps de réponse moyen.

Une étude expérimentale de l'algorithme de réservation sera présentée dans la troisième partie de ce travail.

6.42 Gestion de la mémoire physique

6.421 Organisation générale

On s'intéresse ici à l'ensemble des mécanismes qui permettent de matérialiser sous forme de mémoire physique les mémoires virtuelles des usagers. En fait, le schéma d'adressage exposé au § 6.2 implique que toute allocation de page à un usager se fasse pour le compte d'un article de fichier couplé à une page virtuelle de cet usager.
C'est donc en fait la réalisation de la correspondance entre une information et son support (le couple article de fichier-page physique) qui va être étudiée ici.

On peut distinguer trois étapes dans cette réalisation :

- détection de l'absence d'un article de fichier en mémoire physique ("défaut de page")
- allocation d'une page physique à un article de fichier reconnu manquant
- transferts d'articles de fichier entre mémoire principale et mémoire secondaire.

Le temps de transfert d'une information depuis la mémoire secondaire est très grand par rapport à son temps d'accès en mémoire principale (rapport de l'ordre de 10^4). L'objectif poursuivi dans la mise en oeuvre de l'allocation de pages est la réduction du temps total utilisé pour les transferts entre les deux niveaux de mémoire. Il en résulte un certain nombre de conséquences pratiques :

- moment de l'allocation :

En ce qui concerne l'instant où l'allocation est faite, on a le choix entre le préchargement, à l'entrée d'un usager dans l'état courant, des pages dont l'utilisation est jugée probable, et le chargement "à la demande".

Le chargement à la demande minimise le nombre de transferts, mais pas nécessairement le temps total d'exécution d'un programme. L'avantage du préchargement est de permettre une meilleure utilisation de l'unité centrale, en réduisant la probabilité de défaut de page (chaque défaut de page entraîne un changement d'allocation du processeur). Toutefois l'amélioration n'est réelle que si le système est peu chargé (usagers courants peu nombreux). Par ailleurs, compte tenu du matériel disponible, le coût d'une estimation raisonnable de l'ensemble de pages à précharger est élevé, (il faudrait au minimum un compteur par page virtuelle, mis à jour par programme). On a donc écarté le préchargement, sauf éventuellement pour les pages d'utilisation "certaine".

- élimination des transferts inutiles

- dans le sens mémoire-disques ("vidage")

Seules seront recopiés sur disques les articles de fichier qui ont été modifiés depuis leur dernier chargement. Les pages dont le contenu a été modifié devront donc pouvoir être distinguées des autres, qu'elles soient allouées à un usager ou libres. (on notera l'absence de mécanisme câblé facilitant cette séparation).

Le mécanisme du couplage implique que tout article de fichier ait un support unique à tout instant, (mémoire centrale ou disque) et impose donc la "récupération" en mémoire des articles de fichiers contenus dans des pages libérées.

Le contenu des pages libérées devra donc être préservé le plus longtemps possible pour augmenter la probabilité de récupération. En conséquence,

- seront réallouées en priorité les pages libres contenant des articles de fichier intacts (pas de vidage nécessaire),

- pour les pages contenant des articles de fichier modifiés (pages dites "quasi-libres"), le vidage sera retardé jusqu'au moment de la réallocation.

La réallocation d'une page quasi libre étant coûteuse, il est souhaitable d'avoir parmi les pages disponibles une fraction élevée de pages libres. Un moyen d'y parvenir est d'utiliser les périodes d'oisiveté du canal mémoire-disques pour vider "préventivement" des articles de fichiers contenus dans des pages quasi-libres, les transformant ainsi en pages libres. Cette pratique constitue une exception à la dernière règle énoncée.

- dans le sens disque-mémoire ("chargement")

Toute page transférée (nécessairement à la suite d'un défaut de page) devra être effectivement utilisée par l'usager demandeur. Cela implique que l'usager ne quitte pas l'état courant entre l'instant de la demande et l'instant de l'exécution.
Or, tout défaut de page est occasionné par l'exécution d'un processus, alors que le départ de l'état courant dépend de facteurs intéressant l'usager (catégorie).
La procédure suivante permet de satisfaire à la condition requise :

- les demandes de chargement ne se font pas individuellement par processus mais globalement pour tous les processus d'un même usager. On donnera le nom de requête à une telle demande groupée.

- toute requête émise par un usager quittant l'état courant est différée jusqu'au retour de l'usager dans cet état.

Ce groupement des demandes présente par ailleurs les deux avantages suivants :

- un même article de fichier demandé par plusieurs processus d'un même usager ne fait l'objet que d'une demande de chargement (les demandes multiples étant détectées avant toute action à la constitution de la requête)

- l'optimisation de l'ordre des transferts est plus efficace avec des demandes groupées (cf § 6.441).

6.422 Structures de données

Les structures de données introduites pour mettre en oeuvre les algorithmes d'allocation de mémoire ont été établies en fonction des considérations qui précèdent. La structure principale est la file de pages physiques, qui regroupe des pages de même appartenance. Toute page physique appartient à une telle file. Ces files sont implantées sous forme d'anneaux bilatères, additions et suppressions d'éléments pouvant se faire à n'importe quelle position.

Un recensement des pages physiques permet de distinguer les pages résidentes, les pages attribuées à des usagers pendant la durée de leur existence (pages de garde par exemple), et enfin les pages mobiles qui peuvent être libres ou allouées à des fichiers d'usagers.

Parmi les pages mobiles, une seconde classification permet de distinguer les pages intactes des pages modifiées. Une page "intacte" est caractérisée par l'existence d'une image en mémoire secondaire, ou est vierge (son contenu est sans signification).

Le passage d'une page d'un état d'allocation à un autre est une opération qui peut être longue, et qu'on ne peut en tout cas rendre ininterruptible. Pour garder des informations cohérentes pendant ces phases transitoires, on introduit la notion de page en transit : une page est dite en transit si une opération de transfert la concernant est en cours.

Au total, on a défini six états Eij pour les pages physiques mobiles, avec

$$\begin{cases} i = \text{intacte} \mid \text{modifiée} \mid \text{en transit} \\ j = \text{allouée} \mid \text{non allouée.} \end{cases}$$

Les différentes files de pages sont définies comme suit :

- pour chaque fichier $\begin{cases} \text{une file des pages intactes} \\ \text{une file des pages modifiées} \end{cases}$ allouées à ce fichier

- une file des pages libres (libres intactes)

- une file des pages quasi-libres (libres modifiées)
- une file des pages libres en transit (passant de quasi-libres à libres)

La figure 6.6 schématise l'organisation des files de pages :

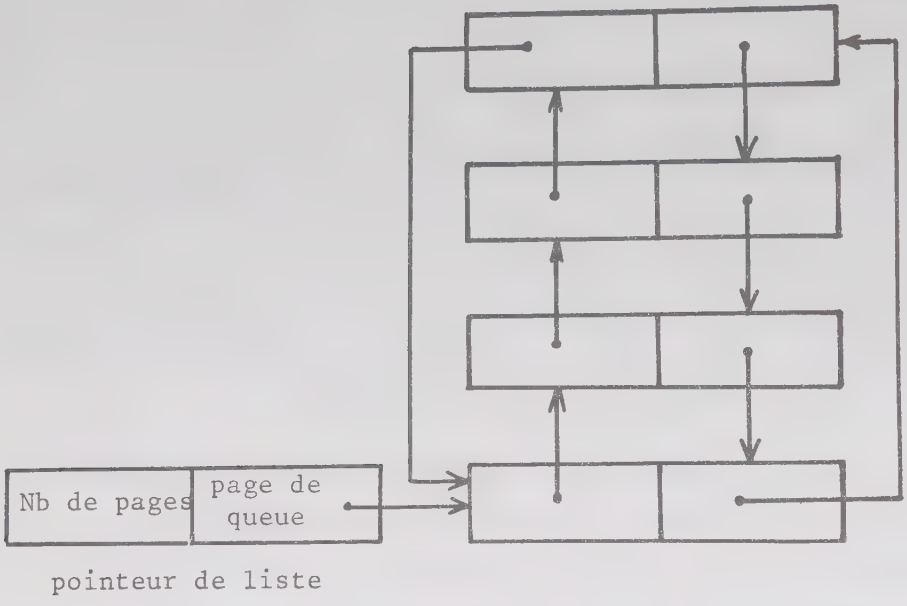


fig. 6. : Organisation des files de pages physiques

6.43 Allocation des pages physiques

Nous décrirons successivement ici les deux premières phases du mécanisme d'allocation :

- la détection du défaut de page
- l'allocation d'une page physique à un article de fichier manquant

6.431 Traitement du défaut de page

L'algorithme de détection du défaut de page utilise les mécanismes de protection de la mémoire virtuelle ; il n'existe pas en effet de mécanisme câblé ad hoc sur le 10070.

Le principe de l'algorithme est le suivant :

A chaque usager (donc à chaque mémoire virtuelle) est attachée une table contenant la protection associée à chacune des pages virtuelles, protection fixée par l'usager, ou "programmée". Lorsqu'un usager est élu, les registres de la mémoire topographiques sont chargés avec la protection '11' signifiant "tout accès interdit". Les programmes des usagers s'exécutant en

mode "esclave avec topographie", toute référence à une page virtuelle provoque un déroutement. Le programme d'analyse du déroutement prévoit alors les cas suivants :

- la protection programmée de la page en cause est incompatible avec le mode d'accès de l'instruction qui a provoqué le déroutement. Il s'agit donc d'une erreur de programmation, qui est signalée à l'usager par le mécanisme standard de traitement des erreurs. Les causes initiales de l'erreur peuvent être diverses : action incompatible avec la protection du fichier auquel est couplée la page, ou défaut de couplage,

- il s'agit d'une première écriture dans une page jusqu'alors intacte (le mécanisme de protection est également utilisé pour traiter ce cas). La protection effective de la page est alors modifiée, (l'écriture est désormais permise) de façon à éviter le déroutement lors des écritures ultérieures ; la page est transférée dans la file des pages modifiées du fichier auquel elle appartient,

- enfin, il s'agit effectivement d'un défaut de page. Deux cas sont encore possibles :

  - la page a été libérée, mais se trouve encore physiquement en mémoire : sa récupération est obligatoire, et une procédure est appelée à cet effet,

  - la page est effectivement absente de la mémoire physique. Son numéro est noté, en vue de la préparation ultérieure d'une requête de chargement ; quant au processus ayant subi le défaut de page, il est bloqué en attendant la satisfaction de la requête ; le mécanisme de ce blocage et du réveil ultérieur sera discuté au chapitre 7.

Les renseignements nécessaires à la préparation de la requête sont accumulés, au fur et à mesure des déroutements, dans la Table des Pages Demandées (TPD) de l'usager. Ces renseignements comprennent :

- le numéro virtuel VTPD de la page manquante
- la nature (lecture, écriture, exécution) de l'accès à la page
- le repérage de l'article demandé (numéro de fichier FTPD, numéro d'article ATPD), ainsi que la protection programmée PFTPD.

Nous allons décrire schématiquement l'algorithme du traitement du défaut de page.

Faisons deux remarques préliminaires :

- le mécanisme câblé du déroutement sur la 10070 ne donne pas directement accès à l'adresse virtuelle dont le calcul a provoqué le déroutement. Cette

adresse doit être déterminée par programme, en analysant l'instruction ayant donné lieu au déroutement ; cette analyse peut être assez complexe, en raison des nombreux modes possibles de génération d'adresse : indirection, chaine d'instruction 'EXECUTER', etc...
Il s'agit là, à notre avis, d'un défaut de conception du calculateur : l'adresse recherchée est en effet disponible dans un registre interne et il serait possible d'y accéder par lecture directe.

- Plusieurs déroutements de causes différentes ont été regroupés à la même adresse (#40) : il est donc nécessaire de déterminer au début de l'analyse, la cause du déroutement. Tel est le sens du test exécuté après le calcul de l'adresse.

L'algorithme du déroutement est donné à la page suivante.

```
begin
calcul du numéro de la page virtuelle V où s'est produit le déroutement ;
XUS := n° d'usager actif ;
if violation de protection virtuelle then
      begin
      F  := FTPV.XUS ;        /* numéro du fichier                    */
      P  := PTPV.XUS ;        /* numéro de page physique              */
      PE := PETPV.XUS(V) ;    /* protection effective                 */
      PF := PFTPV.XUS(V) ;    /* protection programmée                */
      if (PE = '10') & (PE = PF) then
           erreur (1)
      else
           begin
           if (PE = '01') & (PF ='00') then
                 begin     /* première écriture          */
                 if E (P) ≠ 1 then
                       begin
                       transférer (P, QPI(F), QPE(F)) ;
                       E (P) := 1 ;
                       end ;
                 PETPV (V) := PFTPV (V) ;
                 retour (1) ;
                 end
            else
                 begin          /* (PE = '11') & (PF ≠ '11') */
                 if (¬ écriture) | (PF = 0)  then
                       begin
                       préparation de la TPD.XUS
                       retour (2)
                       end
                 else erreur (2)
                 end
      end
end
```

Déroutement pour défaut de page

Récupération des pages

Lorsqu'au cours du traitement du défaut de page on trouve en mémoire l'article recherché (c'est-à-dire que le champ PTIF de la Table d'Implantation du Fichier n'est pas nul), alors sa récupération est obligatoire, en raison de l'unicité de la localisation des articles. La procédure de la récupération proprement dite dépend :

- de l'état de la page récupérée
- du mode d'accès à cette page.

Pour l'état, les différentes possibilités sont les suivantes :

			A	E	T
- page déjà allouée à un fichier	chargée	intacte	1	0	0
		modifiée	1	1	0
	en transit	intacte	1	0	1
		modifiée	1	1	1
- page non allouée	libre		0	0	0
	quasi-libre		0	1	0
	en vidage anticipé		0	1	1

Examinons-les successivement :

La récupération d'une page déjà allouée et chargée signifie qu'un même article de fichier se trouve couplé à deux pages virtuelles différentes du même usager. Aucune action n'est nécessaire dans ce cas, sauf s'il s'agit d'une première écriture dans une page initialement intacte. Les actions sont décrites dans la procédure ci-après :

```
procédure récupérer page allouée (P, F, écriture) ;
    if (P intacte) & (écriture) then
       begin
       transférer (P, QPI [F], QPE [F]) ;
       E [P] := 1                          /* marquer P comme écrite */
       end ;
```

La récupération d'une page non allouée ne pose pas de problème particulier si la page n'est pas en transit ; suivant l'état initial de la page, et la nature de l'accès requis, on aura :

```
procédure récupérer page libre (P, F, écriture)
     if écriture then
          begin
          transférer (P, QPL [F], QPE F ) ;
          E[P] := 1
          end
     else transférer (P, QPL[F], QPI[F]) ;


procédure récupérer page quasi-libre (P, F) ;
     begin
     A[P] := 1 ;
     transférer (P, QPQL[F], QPE[F] ) ;
     end
```

Considérons à présent le cas où la page récupérée est en vidage anticipé. Une telle page est en instance de transfert vers le disque, et sa version à jour se trouve en mémoire centrale. Tant que le transfert n'est pas exécuté (la version sur disque étant alors à jour) la copie en mémoire centrale ne doit pas être perdue. Cela impose de transférer la page récupérée dans la liste des pages modifiées de son fichier (en effet, les demandes de transfert étant réordonnées, on ne peut pas garantir qu'un nouvel article ne sera pas rechargé dans la page avant l'exécution du vidage). On a donc la procédure suivante :

```
procédure récupérer page en vidage (P, F) ;
     begin
     T [P] := 0 ;
     A [P] := 1 ;
     transférer (P, QPV[F], QPE[F]) ;
     end
```

Il reste à examiner le cas où on tente de récupérer une page allouée en transit. Ce cas ne peut concerner qu'une page de fichier partageable dont le chargement a été demandé pour le compte d'un autre usager. La récupération est impossible tant que la page n'est pas effectivement en mémoire. La solution dans ce cas consiste à ne rien faire : le déroutement se reproduira jusqu'à ce que l'article soit chargé. Une page de fichier partagé reste en mémoire tant que le fichier est ouvert pour au moins un usager courant : la récupération aura donc effectivement lieu.

La procédure de récupération s'écrit finalement :

```
procédure récupérer page  (P, V, F, écriture)
     begin
     PTPV [V] := P ;
     if A [P] = 1 then
          récupérer page allouée (P, F, écriture)
else
     if PEQPV then
          récupérer page en vidage (P, F)
else
     if PEQPQL then
          récupérer page quasi libre (P, F)
else
     if PEQPL then
          récupérer page libre (P, F, écriture)
     PETPV(V) := if PFTPV(V) ≠ 0 then PFTPV(V)   /* écriture interdite */
                 else
                 if écriture then '00'
                             else '01' ;
end
```

6.432 Préparation de la requête

La préparation de la requête pour un usager est le préliminaire de son entrée dans l'état suspendu. Cette phase comprend :

- l'allocation effective de pages physiques aux articles de fichiers reconnus manquants (et non récupérés au cours du déroutement)

- la préparation des transferts entre mémoire centrale et disque nécessaires pour le chargement de ces articles.

Les renseignements nécessaires, collectés au fur et à mesure des défauts de page des processus de l'usager, sont rassemblés dans la Table des Pages Demandées (TPD) de l'usager. La composition de cette table a été indiquée au § 6.431.

La communication avec le processus chargé de l'exécution des transferts se fait via la Table de Préparation des Requêtes (TPR), commune à tous les usagers, où les requêtes sont déposées.

Pour éviter tout transfert inutile, les demandes d'articles de fichier enregistrées dans la TPD ne sont transformées en requête que si l'usager peut effectivement passer à l'état suspendu.

La préparation de la requête se fait donc suivant un algorithme à deux passes.

Première passe

Au cours de cette passe, qui est essentiellement une phase de contrôle, on fait les actions suivantes :

- détection des demandes multiples du même article de fichier par des processus différents : ces demandes ne doivent se traduire que par une seule demande de chargement.

- détection du dépassement du nombre de pages permis par la catégorie de l'usager. Dans ce cas, l'usager est transféré à l'état prêt, et aucune requête n'est préparée.

- détection de la création de nouveaux articles de fichiers.

Cette dernière action appelle quelques commentaires :

Quand un fichier est ouvert en écriture, un usager peut désirer créer de nouveaux articles : pour cela, il lui suffit de coupler l'article souhaité à l'une quelconque de ses pages virtuelles, et de tenter d'écrire dans cette page. La création proprement dite de l'article n'a lieu qu'au moment de l'accès à la page, en application du principe qui consiste à n'entreprendre chaque action que le plus tard possible. Les actions correspondantes peuvent comprendre :

- l'allocation d'espace sur disque : cet espace est alloué par "granules" de plusieurs pages ; la création d'un nouvel article peut donc entraîner l'allocation d'un nouveau granule.

- l'agrandissement de la Table d'Implantation du Fichier.

- l'allocation d'une page physique "vierge" destinée à recevoir l'article.

Les deux premières actions peuvent donner lieu à échec (pour manque de place) : dans ce cas également, la préparation de la requête est abandonnée, l'usager est transféré à l'état prêt, et un message signale l'échec de la création d'article.

La procédure ci-après réalise, pour l'usager XUS, les opérations de la première passe de la requête :

```
procédure  requête passe 1 (XUS) ;
     begin
     procédure traiter premier accès ;
          begin
          U(A) := 1 ;              /* nouvel article TIF */
          NA   := 1 ;              /* nouvel article TPD */
          end ;
     procédure traiter accès non initial ;
          begin
          NA := 0 ;
          if fichier partagé then NPC(F) := NPC(F) + 1 ;
          end ;                    /* compte des pages partageables */

     for X∈TPD.XUS do   /* traiter toutes les pages de la requête */
          begin
          boolean accès en écriture ;
          accès en écriture := (PFTPD(X) = '0') ;
          détection des erreurs ;
          if accès en écriture then
               begin
               if débordement de la TIF de F then
                    allouer disque et espace pour TIF
               else
                    if U(A) = 0 then traiter premier accès
                                else traiter accès non initial ;
               end
          else
               traiter accès non initial ;
     end
```

Les erreurs détectables au cours de cette passe sont :

- la tentative d'accès à un article qui n'existe pas
- la tentative d'accéder autrement qu'en écriture à un article nouvellement créé
- l'impossibilité d'agrandir le fichier (défaut de place en mémoire pour la TIF, ou sur disque pour le fichier).

A la fin de la première passe, on dispose du nombre total de pages distinctes demandées pour les fichiers non partagés, et on peut ainsi contrôler le dépassement de la taille correspondant à la catégorie de l'usager.

Deuxième passe

La seconde passe de la préparation de la requête comprend

- l'allocation effective de pages physiques pour les articles de fichiers manquants
- la constitution de la requête proprement dite, y compris la recopie sur disque des pages quasi-libres éventuellement allouées.

Les contrôles effectués pendant et après la première passe garantissent le déroulement sans incident de la seconde passe.

Les pages à allouer sont prélevées successivement dans deux listes :

- liste des pages libres : le contenu de la page n'a pas à être préservé ; le chargement éventuel d'un nouvel article est dit inconditionnel.
- liste des pages quasi-libres : le contenu de la page doit être préservé avant toute nouvelle utilisation, le chargement éventuel d'un nouvel article est dit conditionnel.

Pour permettre au processus chargé du transfert d'exécuter les opérations de vidage avant les chargements conditionnels, une information supplémentaire a été introduite dans la TPR : pour chaque entrée, un bit C indique si le transfert est conditionnel ou non.

Les actions entreprises à l'allocation sont les suivantes :

allocation d'une page libre

- rendre inaccessible l'ancien contenu de la page : on remet pour cela à zéro le numéro de page physique PTIF de l'article dans la TIF de son fichier,
- mettre à jour la TIF correspondant au nouveau fichier :
  PTIF prend une valeur égale au numéro de la page allouée,
- mettre à jour la TPP,
- faire passer la page dans l'une des listes (pages intactes ou pages modifiées) du nouveau fichier.

allocation d'une page quasi-libre

Pour maintenir des informations toujours exactes dans la TPP, les actions sont effectuées en trois temps :

- à l'allocation de la page,
- à la fin du vidage de l'ancien contenu de la page,
- à la fin du chargement du nouvel article.

On traitera des deux derniers dans le paragraphe consacré à l'acquittement de la requête.

Au moment de l'allocation, on se borne aux actions suivantes :

- rendre inaccessible l'ancien contenu de la page,
- mettre à jour la TIF du nouveau fichier,
- mettre à jour la TPP.

En raison de la complexité de la procédure d'acquittement correspondante, (cf.[M2] , § 6.122) il a été décidé de ne pas allouer de pages en vidage anticipé.

La procédure "allouer page", décrite à la page suivante, alloue une page physique à un article de fichier. On désigne par LISTE la liste de pages destinataire, par C la marque de chargement conditionnel, par D l'adresse sur disque de l'ancien article contenu dans la page allouée. (D = 0 si la page est libre). P est le numéro de la page physique allouée.

```
procédure allouer page (LISTE, P, D, C) ;
    begin F := FTPP[P] ; A := ATPP[P] ;
    if NPL ≠ 0 then
        begin                          / * page libre                    * /
        P := tête (QPL) ;
        transférer (P, QPL, LISTE) ;
        C := 0 ;
        D := 0 ;
        end
    else
    if NPQL ≠ 0 then
        begin                          / * page quasi-libre              * /
        P  := tête (QPQL) ;
        transférer (P, QPQL, LISTE) ;
        C := 1 ;
        D := DTIF.F[A]
        end
    else erreur ;
    if F ≠ 0  then                     / * mise à jour ancien fichier de P  * /
        begin
        NPF[F]:= NPF[F]- 1 ;
        PTIF.F[A]:= 0 ;
        end
    end
```

6.44 Transferts d'information

Les requêtes constituées par les différents usagers sont regroupées dans la Table de Préparation des Requêtes (TPR) et traitées ensemble. Ce regroupement des requêtes est dicté par un souci d'optimisation du débit des transferts ; corrélativement se pose le problème de signaler aussitôt que possible à chaque usager la fin d'exécution de sa requête propre ("acquittement" de la requête).

Nous décrirons successivement les mécanismes d'exécution et d'acquittement des requêtes.

6.441 Exécution des requêtes

Etant donné un ensemble de demandes de transfert de pages, provenant d'une ou de plusieurs requêtes, le problème posé est d'obtenir un débit maximal

sur un grand nombre de pages, ou encore de minimiser le temps moyen de transfert d'une page.

Examinons les contraintes du problème :

- lors de la préparation de la requête, l'emplacement sur disque de toutes les pages à vider est déterminé par l'implantation du fichier correspondant. Il n'est donc pas possible d'utiliser les algorithmes d'optimisation fondés sur la détermination "au dernier moment" de l'emplacement des pages à copier sur disque en fonction des emplacements libres.

On traitera donc de manière uniforme toutes les demandes de transfert indépendamment de leur sens.

- l'algorithme d'optimisation des transferts est fondé sur un réarrangement de l'ordre de ces transferts à partir de l'ordre chronologique initial. Toutefois, une contrainte est imposée à ce réarrangement par l'unicité de la localisation des articles de fichier. En effet, lorsqu'un article de fichier est vidé sur disque par suite de l'allocation de la page qu'il occupe à un autre article, le vidage doit effectivement être réalisé avant tout autre chargement du premier article demandé postérieurement, sous peine de charger une version périmée. Montrons sur un exemple qu'un tel cas peut se présenter : soit en effet un usager U_1 possédant une page P (modifiée) contenant un article de fichier (A, F). Considérons la séquence suivante :

- passage de P dans la liste des pages quasi-libres (après blocage de U1)
- allocation de P à un article de fichier (B, G) pour l'usager U_2
- demande de vidage de P (A, F)
- passage de U_1 à l'état courant
- demande de chargement de la page Q (A, F).

Si les deux demandes de transfert font partie de la même requête, et si l'algorithme d'optimisation ne les traite pas dans l'ordre chronologique, alors on chargera Q avec une version périmée de (A, F). La contrainte à respecter est donc la suivante : les demandes de transfert relatives à une même page de disque (ou encore à un même article de fichier) doivent être exécutées dans leur ordre chronologique d'arrivée.

Rappelons les caractéristiques techniques de la mémoire secondaire :

Il existe trois disques gérés par un contrôleur unique.

Chaque disque comprend 512 pistes de 16 secteurs, numérotés de 0 à 15. Le secteur 15 de la piste K et le secteur 0 de la piste K + 1 sont considérés comme contigus. Chaque piste est disposée sur une couronne circulaire pourvue

d'une tête de lecture et d'une tête d'écriture. Un secteur comprend 360 octets.

Pour respecteur le découpage en pages, le transfert se fait par blocs de 6 secteurs contigus, correspondant à 540 mots soit un peu plus d'une page de mémoire. De tels blocs de 6 secteurs seront également appelés pages.

Il résulte de cette disposition que les débuts de page pourront se trouver à 8 emplacements de secteurs possibles sur les 16 (voir figure 6.7), ce qui permet de répartir les pages en 8 files (F0 à F7).

File	Secteurs	Débuts des pages
F0	0	0, 8,...., 8K,....
F1	2	3, 11,...., 8K + 3,...
F2	4	6, 14,...., 8K + 6,...
F3	6	1, 9,...., 8K + 1,...
F4	8	4, 12,...., 8K + 4,...
F5	10	7, 15,...., 8K + 7,...
F6	12	2, 10,...., 8K + 2,...
F7	14	5, 13,...., 8K + 5,...

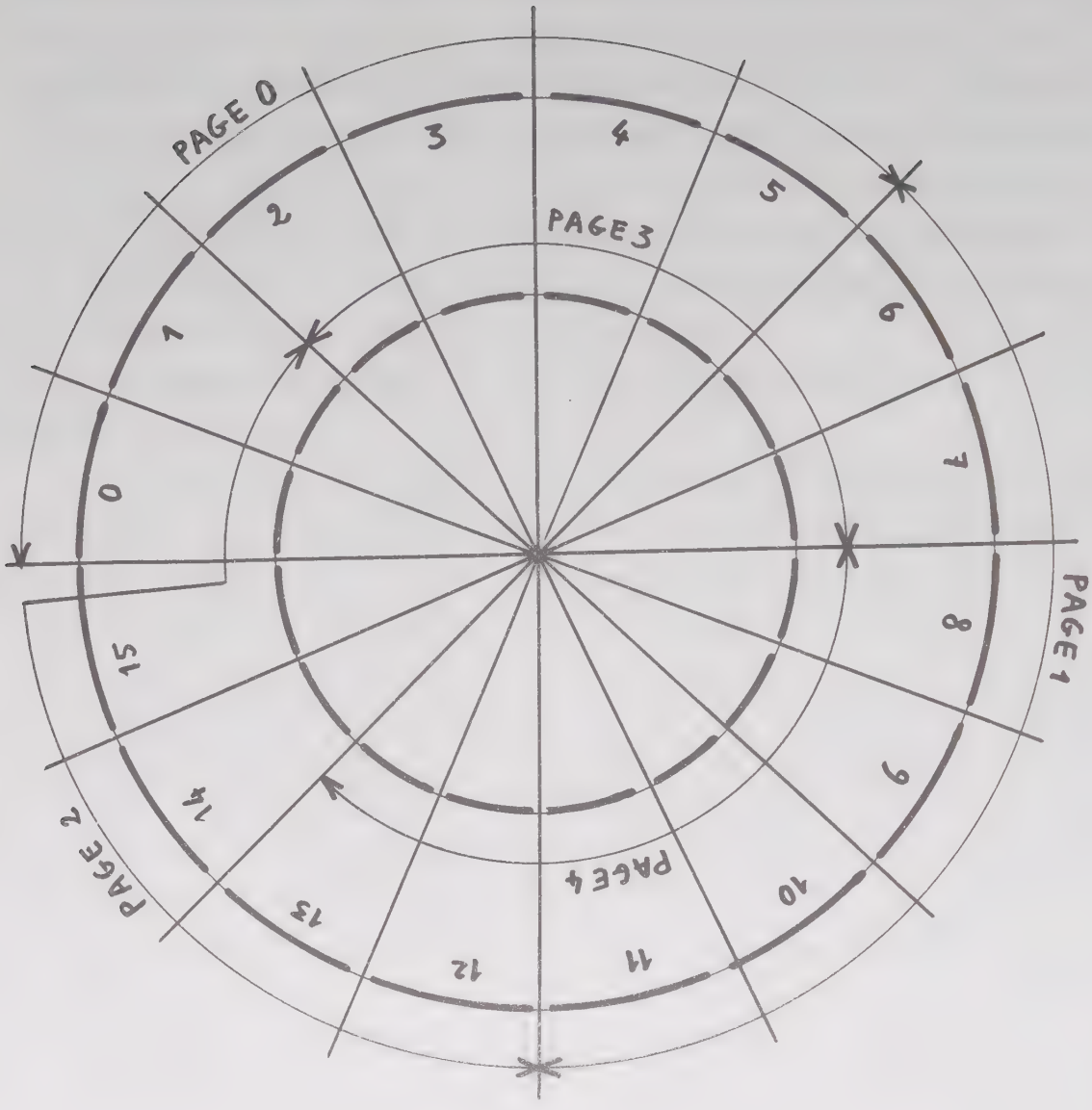


Fig. 6.7 : Schéma de l'organisation des disques en pages

La page de numéro P appartient à la file F_i, où i est le reste de la division de 3P par 8.

A tout instant, il est possible de savoir, grâce à un ordre spécial, sur quel secteur se trouvent les têtes de lecture et d'écriture.

Le temps de transfert (en lecture ou écriture) est de 13 msec par page ; la durée d'un tour de disque est de 33,8 ms.

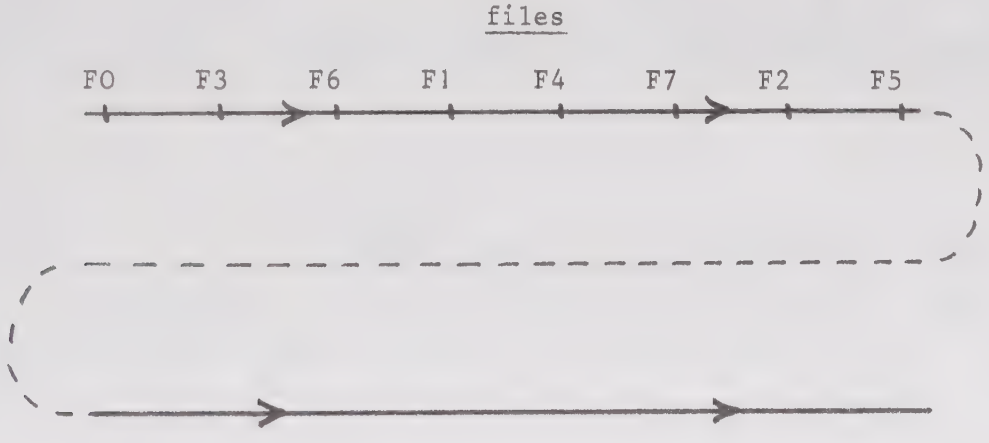
On appellera opération de transfert (ou transfert) l'ensemble des actions déclenchées par une instruction S I O. Aucune opération de transfert ne peut commencer tant qu'un autre transfert est en cours sur la même unité de liaison, plus précisément, tant que l'interruption correspondante n'a pas été

acquittée.

Il en résulte que les demandes parvenant pendant l'exécution d'un transfert doivent être regroupées. Elles seront satisfaites en une opération unique, à la fin du transfert en cours.

Le problème consiste à minimiser le temps total de cette opération. Avant le lancement proprement dit, (instruction SIO) on doit constituer une liste de commandes pour l'unité d'échange. Cela est réalisé au fur et à mesure de l'arrivée des demandes afin de pouvoir lancer le transfert le plus tôt possible. En revanche, la phase d'optimisation tenant compte de la position instantanée du disque ne peut évidemment être faite qu'au dernier moment, c'est-à-dire immédiatement avant le lancement.

L'algorithme d'optimisation utilisé consiste à réordonner les pages suivant la file à laquelle elles appartiennent, de façon à transférer le plus de pages possibles à chaque tour du disque. Cela conduit au classement défini par le schéma ci-dessous.



Ce schéma général peut recevoir deux améliorations :

a) En tenant compte de la position instantanée du disque avant le lancement, on peut déterminer par quelle file doit commencer l'exécution du transfert. On gagne ainsi en moyenne un demi-tour de disque (17 ms) par transfert, (fig. 6.8).

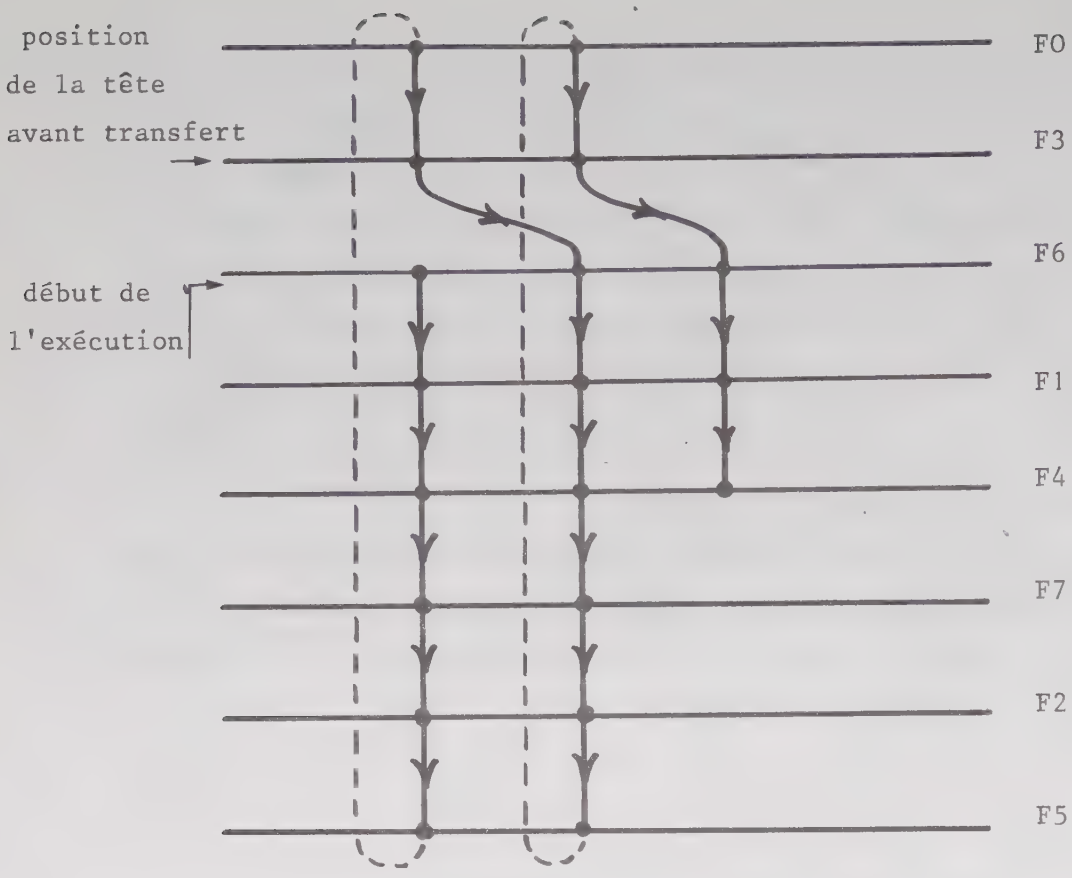


Fig. 6.8 : Organisation des files de transfert

b) On peut gagner une fraction de tour de disque $\left(\frac{1}{8}\right.$ ou $\left.\frac{1}{4}\right)$ en changeant l'ordre des deux dernières pages à transférer lorsque les deux conditions suivantes sont simultanément remplies :

- il y a recouvrement partiel entre la dernière et l'avant dernière page.
- il n'y a pas recouvrement entre l'avant dernière page et l'antépénultième.

On dit qu'il y a recouvrement entre deux pages quand elles contiennent un ou plusieurs secteurs de même numéro. Le recouvrement est total si les deux pages commencent sur le même numéro de secteur, partiel autrement.

En fait, l'amélioration apportée par l'algorithme b) est du second ordre ; on peut le montrer en raisonnant sur le transfert de deux pages (c'est le cas où l'amélioration est la plus importante). En supposant les deux pages réparties au hasard avec une probabilité uniforme, on gagne en moyenne

$$\frac{1}{4} \times \frac{1}{8} \times \frac{1}{8} \times \frac{1}{8} = \frac{3}{64}$$ de tour sur deux tours

soit environ 2,5 %, et beaucoup moins pour une requête plus longue. Seule l'amélioration a) est donc justifiée.

La figure6.9 donne les résultats d'une simulation destinée à mettre en évidence l'optimisation obtenue en réordonnant les demandes de transfert. On a porté, en fonction du nombre de pages transférées, le temps moyen de transfert par page. On voit que le gain obtenu, évidemment nul pour une seule page, croît avec le nombre de pages transférées (d'où l'intérêt de regrouper les requêtes).

Par ailleurs, le temps moyen de transfert sans optimisation tend, quand le nombre de pages croît, vers une limite qu'il est aisé de déterminer théoriquement.

En effet, quand le nombre de pages à transférer est assez grand, les pages peuvent être considérées comme uniformément réparties sur le disque. Le temps moyen de transfert peut alors être calculé comme la moyenne des temps de transfert des 8 pages occupant les 8 positions possibles, ce qui donne :

$$T = \frac{1}{8} \sum_{k=0}^{7} (r_k + t_k)$$

où T est le temps moyen de transfert (limite pour un nombre de pages infini)

r_k le temps d'attente pour une page de la file k

t_k le temps de transfert pour une page de la file k.

La position de la tête étant prise en tête de la file 0, on a :

$$\left.\begin{aligned} r_k &= k\frac{r}{8} \\ t_k &= \frac{3r}{8} \end{aligned}\right\}$$ où r est la durée d'un tour de disque.

$$\text{d'où} \quad T = \frac{3r}{8} + \frac{r}{64}\left(\sum_{K=0}^{7} k\right) = \frac{3r}{8} + \frac{28r}{64} = \frac{13r}{16}$$

valeur que l'on retrouve comme limite asymptotique de la courbe 1 (transfert non optimisé).

Quant à la limite pour $N \rightarrow \infty$ de la courbe 2 (transfert optimisé), elle s'obtient en remarquant que le temps de transfert d'une page est dans ce cas égal au temps de défilement de la page devant la tête de lecture, puisque

toutes les files sont alors pleines. On a donc :

$$T = \frac{3r}{8}$$

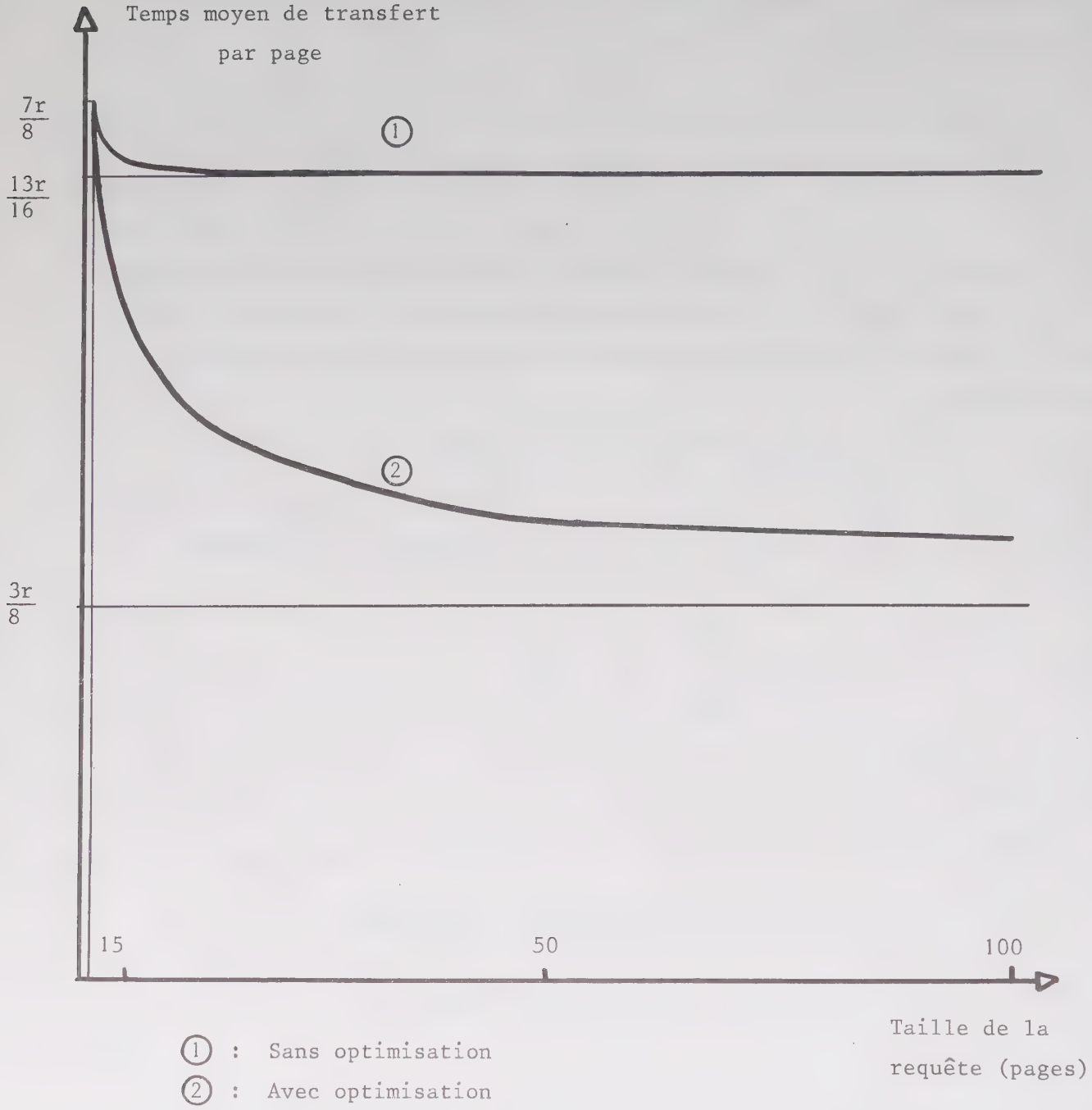


Fig. 6.9 : Temps d'exécution des requêtes.

6.442 Acquittement des requêtes

On désigne sous le nom d'acquittement d'une requête, l'ensemble des opérations consécutives à l'achèvement des transferts spécifiés dans cette requête. Ces opérations sont de deux sortes :

a) tenue à jour des diverses tables d'état des pages

b) actions sur l'état de l'usager concerné.

Nous décrivons ici les algorithmes réalisant les opérations a).

Une requête pour le compte d'un usager comprend 3 types de transferts : chargement inconditionnel, vidages, chargements conditionnels (dans les pages libérées par les vidages). Le problème de la tenue à jour de l'état des pages est essentiellement un problème de sécurité : il importe de connaître à tout instant l'état exact de l'information. Dans le cas des situations transitoires, qui est celui des requêtes, on connaît l'état initial (TPV) et l'état final (TIF) ; il est toujours possible de reconstituer l'état initial tant que l'on se trouve dans l'état transitoire noté dans la TPP. D'après la signification des informations de la TPP, on voit que celle-ci doit être mise à jour entre le vidage d'une page quasi-libre et le changement de l'article destiné à la remplacer.

On est donc conduit à prévoir deux étapes pour l'acquittement d'une requête comportant une partie conditionnelle :

1) exécution des vidages et des chargements inconditionnels

2) exécution des chargements conditionnels

avec un acquittement entre les deux phases, dit acquittement intermédiaire destiné à la mise à jour de la TPP pour enregistrer les résultats de la première phase et préparer la seconde. L'acquittement de la seconde phase est dit acquittement terminal.

Si la requête ne comporte pas de partie conditionnelle, elle se réduit à la première phase et se termine donc après l'acquittement intermédiaire.

Nous donnons ci-dessous la description des deux algorithmes :

Acquittement intermédiaire

Les informations concernant les pages vidées sont remplacées, dans la TPP, par celles concernant les nouvelles pages à charger. La marque de transit est effacée pour toutes les pages transférées. Notons que si une page rendue libre par vidage est affectée à un article nouvellement créé, donc vierge (création ou extension d'un fichier), alors il n'y a pas de transfert,

mais remise à zéro de la page de mémoire correspondante et du bit de transit. Aucune autre opération n'a lieu pour cette page.

```
        procédure acquittement intermédiaire (XUS) ;
            begin
            for (P∈{pages en vidage et en chargement inconditionnel dans
                TPD.XUS}) | (P≠0) do

    begin
    if VP[P]= 0 then
        T[P]:= 0                /* chargement inconditionnel */
    else
    if NA[P]= 1 then            /* nouvel article de fichier */
        begin
        remettre à zéro le contenu de la page P ;
        T[P]:= 0
        end ;
    FTPP[P]:= FTPD[P];
    ATPP[P]:= ATPD[P];
    end
end
```

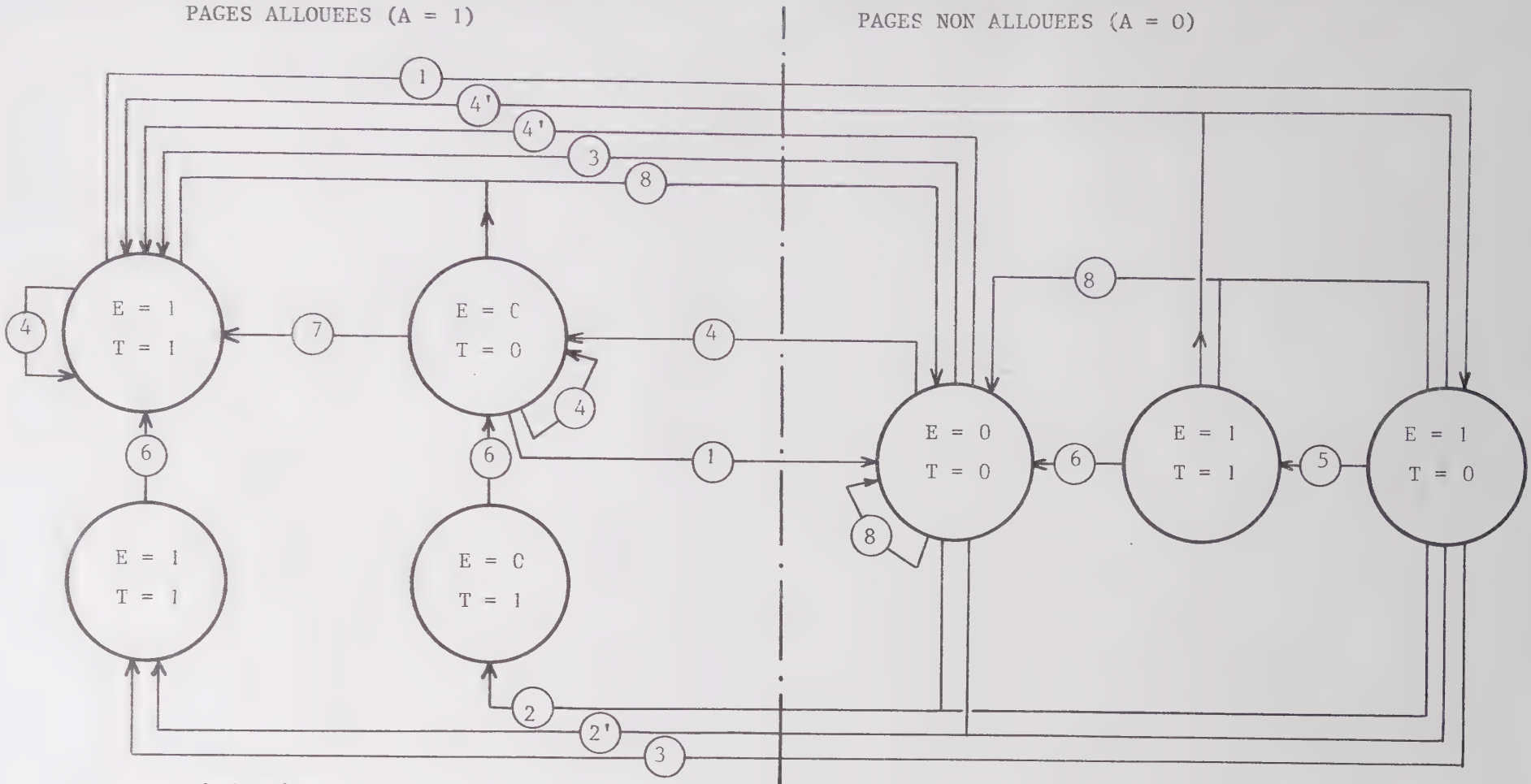
Acquittement terminal

Dans ce cas, l'action se résume à une remise à zéro de la marque de transit (sauf dans le cas d'un nouvel article, où cela a été fait dans l'acquittement intermédiaire)

```
    procédure acquittement terminal (XUS) ;
        begin
        for P∈{pages en chargement conditionnel dans TPD.XUS} do
            begin
            if (VP[P]= 1)&(NA[P]= 0) then
                T[P]:= 0
            end
        end
```

6.45 Récapitulation

Une image synthétique de l'allocation des pages physiques est fournie par le schéma de l'ensemble des transitions possibles entre les états des pages allouables. Ce schéma est représenté à la figure 6.10.



1 Libération

2 Allocation avec chargement sans écriture

2' Allocation avec chargement avec écriture

3 Allocation pour création d'article

4 Récupération

4' Récupération avec écriture

5 Vidage anticipé

6 Acquittement de requête

7 Ecriture dans une page intacte

8 Destruction de fichier

Fig. 6 : Transitions entre les états des pages physiques mobiles

Transition n°	Action	Circonstances
1	libération	fermeture de fichier départ d'un usager de l'état courant
2	allocation avec chargement	demande de page (1)
2'	idem avec accès en écriture	(1)
3	allocation pour un nouvel article	accès (écriture permise) à un article inexistant
4	récupération d'une page non allouée	demande de page
4'	idem avec accès écriture	
5	vidage préventif	disque oisif
6	arrivée d'un article	fin de transfert
7	changement de liste (intacte modifiée)	première écriture dans un article déjà chargé (2)
8	libération avec perte du contenu	desctruction de fichier.

Remarques :

(1) des pages allouées non en transit peuvent être récupérées car plusieurs pages virtuelles peuvent être couplées au même article : seule la première référence donne lieu à un chargement.

(2) lorsque le premier accès à un article non chargé est une écriture, la page correspondante est directement placée dans la liste des pages modifiées. La transition 7 ne concerne donc que des pages déjà chargées pour lecture en exécution.

(3) comme les articles des fichiers partagés ne sont accessibles

qu'en lecture ou en exécution, le diagramme de transition entre états de pages allouées à un fichier partagé états de pages allouées à un fichier partagé est un sous-ensemble de ce schéma général.

Chapitre 7

ORGANISATION DU SYSTEME EN PROCESSUS PARALLELES

7.1 Principes de la décomposition

On a décrit dans le chapitre précédent les algorithmes d'allocation des ressources aux usagers dans le système ESOPE. L'objet de ce chapitre est de montrer comment ces algorithmes ont été mis en oeuvre sous la forme d'une famille de processus parallèles, achevant ainsi la description du système ESOPE.

Dans la description d'une activité sous forme de processus parallèles, le but poursuivi est double :

- clarifier la description
- exploiter au mieux le parallélisme.

Une telle décomposition comporte toujours une certaine part d'arbitraire. On peut toutefois faire deux remarques élémentaires :

- une activité qui met en jeu n processeurs fonctionnant en parallélisme réel comporte au moins n processus,

- deux processus qui ne sont jamais actifs simultanément peuvent être considérés comme parties d'un seul processus, si un lien logique entre eux le justifie.

L'étape préliminaire de la décomposition consiste à analyser les mécanismes câblés d'activation et de transfert du contrôle. Sur la 10070, on dispose de deux tels mécanismes : le déroutement et l'interruption. Dans cette analyse, on doit explicitement tenir compte du fait que la machine est mono-processeur.

Un déroutement qui se produit lors de l'exécution d'un programme est provoqué par une cause dépendant du programme lui-même ou de l'environnement dans lequel il s'exécute. Par exemple :

- un déroutement pour instruction inexistante est provoqué par une caractéristique de la procédure,

- un déroutement pour adresse inexistante, ou pour erreur dans une opération arithmétique, ou pour débordement d'une pile est provoqué par des caractéristiques de la procédure et/ou des données,

- un déroutement pour appel-superviseur est explicitement prévu dans la procédure. Il n'est pas fondamentalement différent d'un branchement,

- un déroutement pour violation de protection de mémoire est provoqué par une caractéristique de l'environnement dans lequel le programme s'exécute.

Une interruption, au contraire, est déclenchée par une cause extérieure au programme qui s'exécute. Toutefois, une analyse plus fine permet de distinguer deux types d'emploi de l'interruption :

a) interruption déclenchant l'exécution d'un programme différent du programme en cours d'exécution (et sans rapport avec ce dernier).

exemple : interruption signalant la fin d'une entrée-sortie,

b) interruption déclenchée en vue de l'arrêt du programme en cours d'exécution,

exemple : interruption déclenchée par une horloge.

En termes de processus, ces deux opérations s'interprètent comme suit :

- tout programme de déroutement est considéré comme faisant partie du processus dérouté,

- on associe à toute interruption une opération V sur le sémaphore privé d'un certain processus.

On voit que cette interprétation règle le cas des interruptions de type a). Pour traiter également les interruptions de type b), on est amené à inclure dans la primitive V l'allocation du processeur (on a vu que cette inclusion n'était pas obligatoire en monoprocesseur). On dispose alors d'un mécanisme permettant de provoquer par l'opération V l'arrêt effectif du processus en cours : il suffit que le processus débloqué par V précède le processus en cours dans la chaîne d'activation.

7.2 Mécanismes d'allocation du processeur

L'unité centrale est allouée aux usagers pour l'exécution de leurs processus. A un instant donné, sont seuls demandeurs d'unité centrale les processus prêts des usagers éligibles. L'ensemble des processus du moniteur résident est considéré, en ce qui concerne l'allocation de processeur, comme appar-

tenant à un usager unique constamment éligible ; de plus chacun de ces processus a une priorité supérieure à celle de tout processus d'un usager. Enfin, les usagers éligibles sont ordonnés entre eux par leur appartenance à la file C. La réunion de ces conditions induit sur l'ensemble des processus demandeurs d'unité centrale une relation d'ordre total strict : les processus ainsi ordonnés forment ce que l'on appelle la chaîne d'activation :

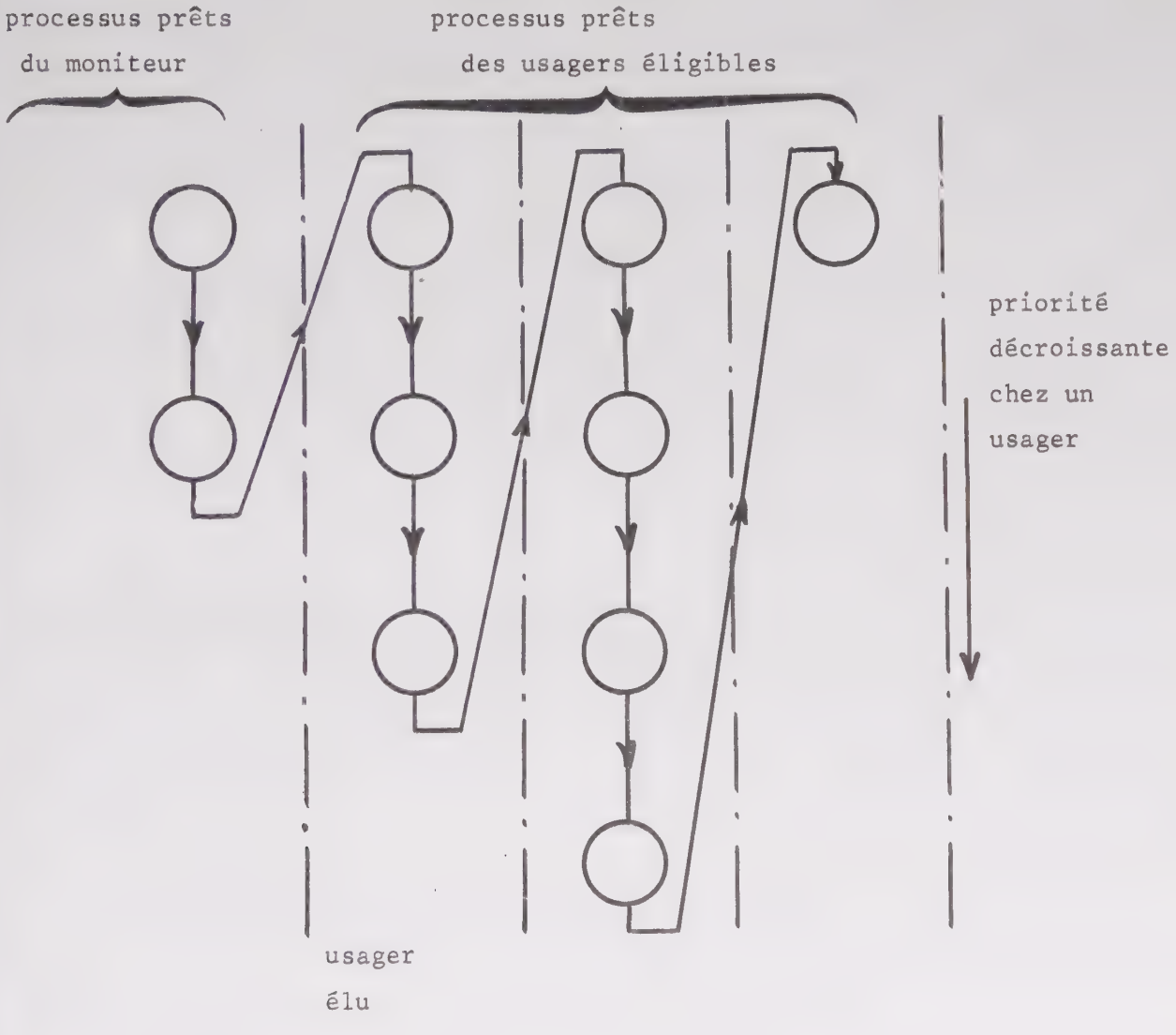


Fig. 7.1 : La chaîne d'activation

La chaîne d'activation à un instant donné définit l'ordre futur d'activation des processus tel qu'il est connu à cet instant ; l'ordre et la composition de la chaîne d'activation varient au cours du temps :

- lorsqu'un processus se bloque, il est transféré de la file des processus prêts de l'usager élu dans une file de sémaphore,
- lorsqu'un processus bloqué d'un usager éligible passe à l'état prêt, il est transféré d'une file de sémaphores dans la file de son usager, à un rang fixé par sa priorité interne,
- lorsqu'un usager devient éligible, la file de ses processus prêts est insérée dans la chaîne d'activation, à un emplacement déterminé par la position de l'usager à l'intérieur de la file C,
- lorsqu'un usager quitte l'état courant, la file de ses processus prêts (si elle n'est pas vide) est retirée de la chaîne d'activation.

A tout instant, le processus élu (celui qui s'exécute sur l'unité centrale) est le premier de la chaîne d'activation. C'est donc soit un processus du moniteur, soit un processus de l'usager élu.

Examinons à présent la relation entre l'allocation de processeur et les primitives de synchronisation de processus.

Un processus qui exécute une primitive P sur un sémaphore, et qui se bloque à la suite de cette action doit libérer le processeur qu'il occupe. La situation est plus complexe dans le cas d'une primitive V qui rend actif un processus précédemment bloqué. En monoprocesseur, il n'est pas obligatoire, du moins en ce qui concerne l'efficacité, de changer l'allocation du processeur, puisque le processus actif peut continuer à l'occuper (la situation est différente en multiprocesseur).
Toutefois, l'utilisation de la primitive V pour la description de l'effet des interruptions nous a conduits à inclure dans V l'opération d'allocation de processeur, et à choisir pour les processus bloqués derrière un sémaphore une structure de queue avec priorité.

L'algorithme des primitives P(s) et V(s), incluant l'allocation de processeur, s'écrit donc (en notant p le processus qui exécute la primitive et Q(s) la file des processus bloqués derrière s) :

```
P(s)   :  begin
          s := s - 1 ;
          if s < 0 then
               begin
               état (p) := bloqué ;
               transférer p de la chaîne d'activation dans la file Q(s) ;
               activer ;
               end
          end

V(s)   :  begin
          s := s + 1 ;
          if s ⩽ 0 then
               begin
               sortir de la file Q(s) le premier processus de cette file
               (soit q) ;
               état (q) := actif ;
               transférer p dans la chaîne d'activation ;
               activer ;
               end
          end
```

La procédure désignée par "activer" est chargée de l'allocation effective du processeur au premier processus de la chaîne d'activation. Son rôle est donc essentiellement de restaurer le vecteur d'état de ce processus, constitué par le contenu des registres généraux du processeur et du double-mot d'état de programme.

7.3 Blocage technologique des processus

Les relations entre processus du moniteur et processus de l'usager devant utiliser les mécanismes standard de synchronisation, il reste à exprimer au moyen de ces mécanismes le blocage technologique d'un processus, c'est-à-dire le blocage pour défaut de page.

La détection du défaut de page utilise un déroutement et l'algorithme correspondant est considéré comme faisant partie du processus dérouté. Comme on l'a vu au § 6.43, il y a trois issues possibles à ce déroutement (première écriture défaut de page, ou erreur). Dans le premier cas, il y a retour au processus dérouté après avoir changé de liste la page objet de l'écriture. Dans les deux autres cas, le processus ne peut continuer son exécution. La

solution choisie est alors de l'obliger à se bloquer derrière un sémaphore. A cet effet, il est prévu pour chaque usager deux sémaphores généraux (inaccessibles directement aux processus de l'usager sauf dans le programme de déroutement), notés S:PAGE et S:ERREUR. Utilisant ces sémaphores, le schéma du déroutement pour défaut de page s'établit comme suit :

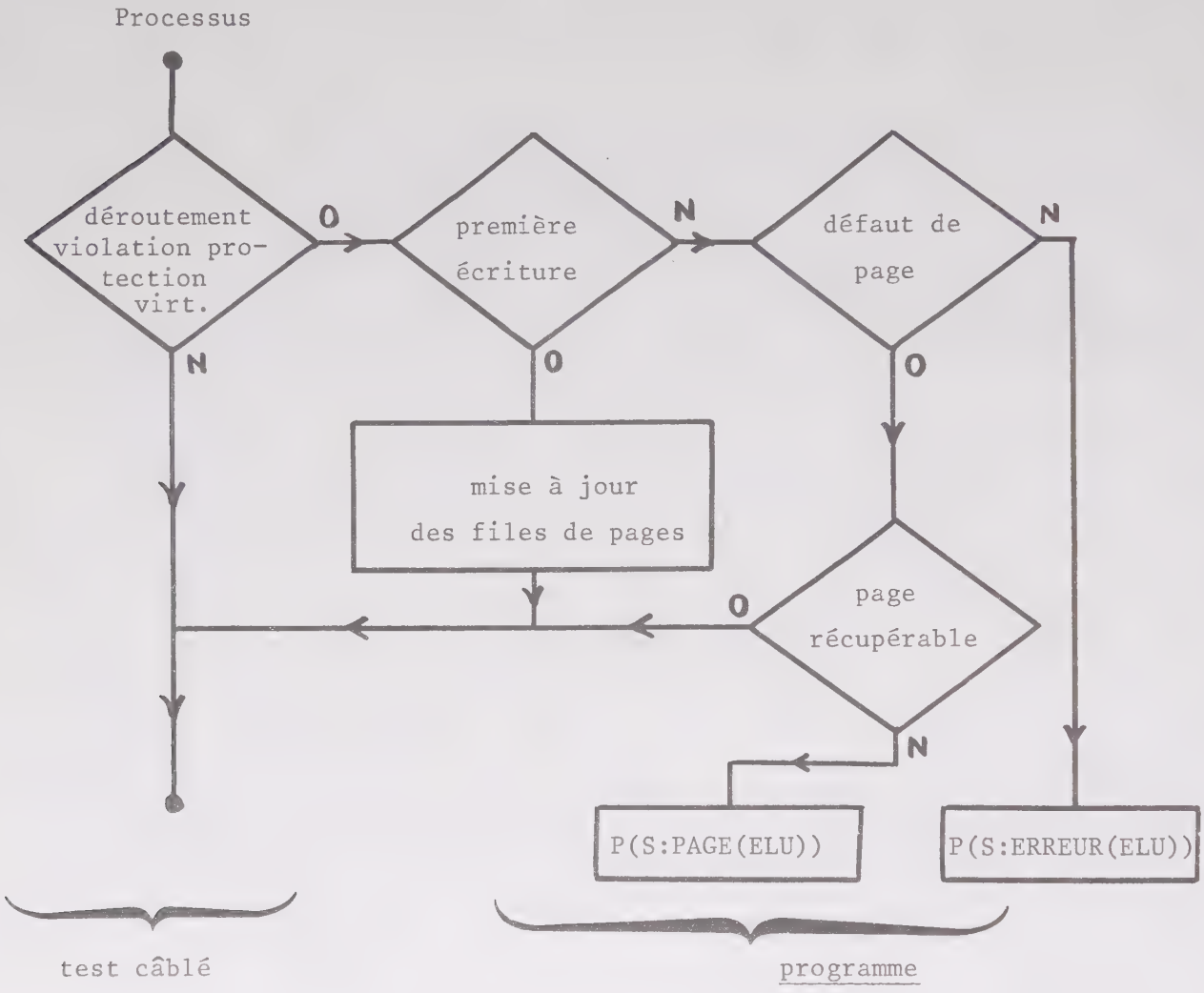


Fig. 7.2 : Détection du défaut de page

Le "test câblé" a lieu en fait à chaque évaluation d'adresse virtuelle dans un programme s'exécutant en mode esclave avec topographie (ce qui est le cas des programmes d'usagers). Avec le schéma indiqué, tout incident concernant la réalisation d'une adresse virtuelle (erreur d'adressage ou défaut de page) se traduit donc par le blocage du processus concerné derrière un sémaphore propre à son usager.

7.4 Processus de l'allocateur

On dispose maintenant de tous les outils nécessaires à la mise en oeuvre sous forme de processus des algorithmes exposés au chapitre 7.

L'association d'un processus distinct à chaque cause d'interruption nous conduit à distinguer au moins les processus suivants dans l'allocateur de ressources :

Interruption	Fonction du processus
Fin de quantum	Changement de l'usager élu
Fin de transfert (physique)	Exécution des requêtes
Fin de transfert (logique)	Acquittement des requêtes
Première entrée sur un terminal	Admission d'un usager

7.41 Changement d'usager élu

Examinons les divers cas de transition dans le cycle des usagers au départ de l'état élu E (en dehors du départ définitif du système, qui sera analysé plus loin)

a) E * $\longrightarrow$ E : fin du quantum d'unité centrale accordé à l'usager,

b) E * $\longrightarrow$ S : tous les processus de l'usager sont bloqués, dont au moins un pour défaut de page,

c) E * $\longrightarrow$ O : tous les processus de l'usager sont bloqués intrinsèquement,

d) E * $\longrightarrow$ P : dépassement d'une limite fixée par la catégorie de l'usager.

. Analysons les causes de ces différentes transitions. On constate que :

- la transition a) est déclenchée par un événement extérieur (interruption d'horloge),
- les transitions b) et c) sont provoquées par le blocage du dernier processus prêt de l'usager élu,
- la transition d) provient soit d'un dépassement de temps de résidence, soit d'un dépassement de nombre de pages occupées. Cette dernière occurence est détectée lors de la préparation de la requête.

On peut ainsi distinguer les diverses manières d'activer le processus CHANGEMENT :

- dans le cas a), l'interruption de fin de quantum provoque une opération V sur le sémaphore privé de ce processus,
- dans le cas d), le test du dépassement du temps de résidence peut être inclus dans le programme fin de quantum : cela revient à mesurer ce temps de résidence avec une précision de l'ordre du quantum, ce qui ne provoque qu'une erreur négligeable,

- les cas b) et c) soulèvent un problème plus délicat. On peut envisager deux solutions :

  1) à chaque blocage d'un processus d'usager, vérifier si ce processus est ou non le dernier de la file des processus prêts de cet usager ; dans l'affirmative, activer le processus CHANGEMENT. Ce test peut être inclus dans la procédure "activer",

  2) inclure en queue de la file des processus prêts de chaque usager un processus supplémentaire toujours prêt. Le rôle de ce processus cyclique, dit FINAL se borne à activer le processus CHANGEMENT ; en d'autres termes, le processus FINAL sert de "marque de fin" dans la file des processus prêts d'un usager. Sa priorité est inférieure à celle de tous les autres processus ; il ne peut donc s'exécuter que s'il est le seul processus prêt de l'usager.

  Les deux solutions ci-dessus sont équivalentes quant à leur effet global. Leurs avantages respectifs sont :

  - pour la solution 1, l'efficacité (le mécanisme d'activation d'un processus est assez lourd, et l'espace (résident) requis pour loger le vecteur d'état d'un processus FINAL par usager est assez important),

  - pour la solution 2, l'élégance et la simplicité conceptuelle : on dissocie, en effet, les opérations de synchronisation de processus et d'allocation de processeur, d'une part, et de gestion des usagers, d'autre part.

La solution 1 est adoptée dans le système par souci d'efficacité ; le schéma présenté ici est conforme, pour la clarté de l'exposé, à la solution 2.

Les fonctions du processus CHANGEMENT jusqu'ici recensées sont :

- dans tous les cas, changer l'usager élu,
- dans le cas b) (transition E $\longrightarrow$ S) se charger de la préparation de la requête.

Remarquons que le processus CHANGEMENT, en raison du cas d'activation a), est activé au moins une fois par quantum. D'où l'idée de lui confier, outre les fonctions ci-dessus, diverses tâches dont l'accomplissement peut sans dommage subir un retard de l'ordre de grandeur du quantum. Ces tâches sont :

- la détection du dépassement du temps de résidence (déjà mentionnée),
- la détection des usagers oisifs dont au moins un processus a été débloqué, et le transfert de ces usagers dans la file des prêts (transition O $\longrightarrow$ P). Ce transfert périodique évite de faire un test à chaque activation de processus.

Enfin, les transitions E$*\longrightarrow$ O et E$*\longrightarrow$ P font sortir un usager de la file C :

Il y a donc libération de mémoire, et on doit appliquer l'algorithme d'admission des usagers prêts dans la file C (transition P ⟶ C).

Dans le cas où aucun candidat n'est éligible, le système entre dans un état d'inactivité, dont il peut sortir dans 3 circonstances :

- acquittement d'une requête, rendant éligible un usager suspendu,
- activation d'un processus bloqué chez un usager oisif (la file P étant vide, cet usager va entrer dans l'état éligible),
- entrée d'un nouvel usager dans le système (même remarque que ci-dessus).

Cet état d'attente du système est simplement réalisé de façon "active" par une boucle du processus CHANGEMENT qui scrute cycliquement la file des usagers courants jusqu'à trouver un usager éligible.

La suite des opérations à exécuter par le processus CHANGEMENT peut en définitive se résumer ainsi :

Préparation de la requête si nécessaire ;
Transférer l'usager élu dans la file correspondant à son nouvel état ;
Faire passer (s'il y a lieu) des usagers oisifs dans la file P ;
" " " " " " prêts " " C ;
Scruter cycliquement la file C jusqu'à trouver un usager éligible et faire passer cet usager dans l'état élu ;

Examinons maintenant les opérations à faire lors du passage à l'état élu d'un usager éligible.

On doit :

- charger dans les registres topographiques les protections effectives correspondant à l'état des diverses pages virtuelles du feuillet de l'usager. Pour toutes les pages couplées de l'usager, ces protections reçoivent la valeur '11' (cf algorithme de détection du défaut de page). Pour les pages de communication (pages 0, 1, page de garde), la valeur est '00' (ces pages étant résidentes),

- charger dans un compteur la valeur du quantum alloué à l'usager ; ce compteur est décrémenté à chaque impulsion d'horloge ; son passage par 0 donne lieu à l'interruption de fin de quantum, et active de nouveau le processus CHANGEMENT.

L'algorithme du processus CHANGEMENT s'écrit finalement :

```
processus CHANGEMENT ;

REPOS : P(S:CHANGEMENT)
        XUS := USRELU ;
        if défaut de page then          /* transition  E *--> S  */
             begin
             if dépassement du temps de résidence then
                  begin
                  ETAT (XUS) := 'prêt' ;
                  libérer mémoire (XUS) ;
                  transférer (XUS, QPRET) ;
                  end
             else
                  begin
                  requête passe 1 (XUS) ;
                  if requête vide then
                       begin
                       état (XUS) := 'éligible' ;
                       D(S:PAGE (XUS)) ;
                       end
                    else
                       begin
                       ETAT (XUS) := 'suspendu' ;
                       requête passe 2 (XUS) ;
                       if disque oisif then V(S:TRANSFERT) ;
                       end
                  end
             end
else
     begin                  /* transition E --> O    */
     ETAT (XUS) := 'oisif' ;
     libérer mémoire (XUS) ;
     transférer (XUS, QOISIF) ;
     end ;
/* fin des actions sur l'ancien usager élu  */
```

```
     /*  détermination des transitions possibles O ⟶ P    */
BOUCLE : for X ∈ QOISIF  do
     if nombre processus prêts (X) > 0 then
          begin
          ETAT (X) := prêt ;
          transférer (X,QPRET)
          end ;
     /*  détermination des transitions possibles P ⟶ C    */
     test d'accès à la file C ;
     /*  élection d'un nouvel usager                       */

repeat
     begin
     XUS := SUCUS (XUS) ;
     if ETAT (XUS) = 'éligible' then
          begin
          USRELU := XUS ;
          charger registres topographiques ;
          charger compteur du quantum d'unité centrale ;
          goto REPOS
          end
     end
until exploration complète de la file C ;
goto BOUCLE                /*  boucle d'attente, si aucun usager éligible  */
end
```

7.42 Exécution des transferts

Le processus TRANSFERER-PAGE d'exécution des transferts entre mémoire centrale et disques a pour tâche de satisfaire les requêtes déposées par les usagers. L'existence d'un contrôleur unique impose l'unicité du processus, ainsi que son mode de fonctionnement : chaque transfert porte sur un ensemble de requêtes, dont les pages sont réordonnées comme indiqué au § 7.441 pour optimiser le débit. Les nouvelles requêtes arrivant pendant l'exécution d'un transfert sont réordonnées au fur et à mesure de leur arrivée ; le transfert correspondant a lieu dès la fin de l'exécution du transfert en cours. Le processus est donc cyclique. Il peut se trouver bloqué lorsqu'aucune requête nouvelle n'est arrivée pendant l'exécution d'un transfert : le processus chargé de la préparation de la requête doit donc, dans ce cas, activer explicitement le processus TRANSFERER PAGE.

Les interruptions d'entrée/sortie survenant pendant l'exécution d'un transfert sont analysées et reçoivent selon leur origine le traitement suivant :

- fin de la partie "vidage" d'une requête
- fin de l'exécution complète d'une requête

} activation du processus d'acquittement (cf. § 7.43)

- fin de transfert (ensemble de requêtes) : réactivation du processus de transfert

L'algorithme du processus TRANSFERER-PAGE s'écrit :

```
TRANSFERER-PAGE :

          begin
ATTENTE : P(S:TRANSFERT) ;
          réordonner les requêtes en attente pour
          optimiser le temps de transfert ;
          if pas de requête en attente
          then
               disque oisif := true
          else
               lancer le transfert (SIO) ;
          goto ATTENTE
          end
```

Le processus est activé de deux façons :

- à l'arrivée de la requête (processus CHANGEMENT) :
  `if disque oisif then V(S:TRANSFERT)`
- à la fin de l'exécution du transfert (interruption = V(S:TRANSFERT))

7.43 Acquittement des requêtes

On distingue deux types d'acquittement suivant la nature du demandeur de la requête :

- acquittement d'une requête du système (vidage préventif de pages quasi-libres)
- acquittement d'une requête pour un usager.

On a vu par ailleurs (6.44) que le second cas se subdivise lui-même en deux (acquittement intermédiaire et acquittement terminal).

Dans tous les cas, l'acquittement est déclenché par un signal (interruption) provenant d'un processus 'EXECUTER REQUETE' parallèle aux processus en cours sur l'unité centrale. Les opérations d'acquittement sont logiquement indépendantes de ces processus ; on les a donc mises à la charge d'un processus spécial, 'ACQUITTER REQUETE'.

7.431 Acquittement pour le système

Dans ce cas, (vidage préventif), la seule action requise est la mise à jour des files de pages (les pages en vidage passant dans la file des pages libres).

L'algorithme s'écrit :

```
procédure acquittement pour le système ;
    begin
    for P∈QPV do
        begin
        T(P) := 0 ;
        E(P) := 0
        end ;
    QPL :=  QPL || QPV ;
    NPL := NPL + NPV ;
    NPV := 0
    end
```

7.432 Acquittement pour un usager

Le principe de la décomposition en deux phases des requêtes des usagers, en signalant séparément, la fin des vidages et chargements inconditionnels, a été indiqué au § 6.442, où ont également été décrits le mécanisme de l'acquittement intermédiaire et de l'acquittement terminal.

Les actions à entreprendre sont :

- à la fin d'un acquittement intermédiaire (et s'il reste encore des pages à charger), mettre simplement à jour le compteur de vidage,
- à la fin de l'acquittement de l'ensemble des demandes (qu'il s'agisse d'un acquittement intermédiaire ou terminal), on doit mettre à jour la TPV de l'usager, et rendre éligible l'usager (suspendu) pour lequel s'exécute la requête.

Tous les processus intrinsèquement prêts de l'usager étant alors bloqués derrière le sémaphore S:PAGE(XUS), cette action est réalisée par une primitive D sur ce sémaphore, qui active tous ces processus ; remarquons toutefois que l'emploi d'une primitive D au lieu d'une boucle sur une opération V constitue essentiellement ici une facilité d'écriture et n'a pas de justification intrinsèque.

L'algorithme de l'acquittement d'une requête pour un usager XUS est décrit ci-dessus. On rappelle les notations :

NCC[XUS] : nombre de pages en chargement conditionnel
OUT[XUS] : nombre de pages en vidage

} dans la requête en cours pour XUS

```
procédure acquittement pour l'usager (XUS) ;
     begin
     procédure mise à jour de la TPV.XUS ;
          begin
          for  V∈TPD.XUS do
               begin
               PTPV.XUS [V] := PTPD.XUS [V] ;
               PE.XUS [V] := PETPD.XUS [V] ;
               end
          end
     if NCC[XUS]= 0 then
          begin              /*  pas de chargements conditionnels * /
          acquittement intermédiaire (XUS) ;
          mise à jour de TPV.XUS ;
          D(S:PAGE XUS ) ;
          end
     else
          begin
          if OUT[XUS]≠ 0 then
               begin
               acquittement intermédiaire (XUS) ;
               OUT[XUS]:= 0 ;
               end           /*  doit être suivi d'un acquittement terminal * /
          else
               begin
               acquittement terminal (XUS) ;
               mise à jour de la TPV.XUS ;
               D(S:PAGE XUS ) ;
               end
          end
     end
```

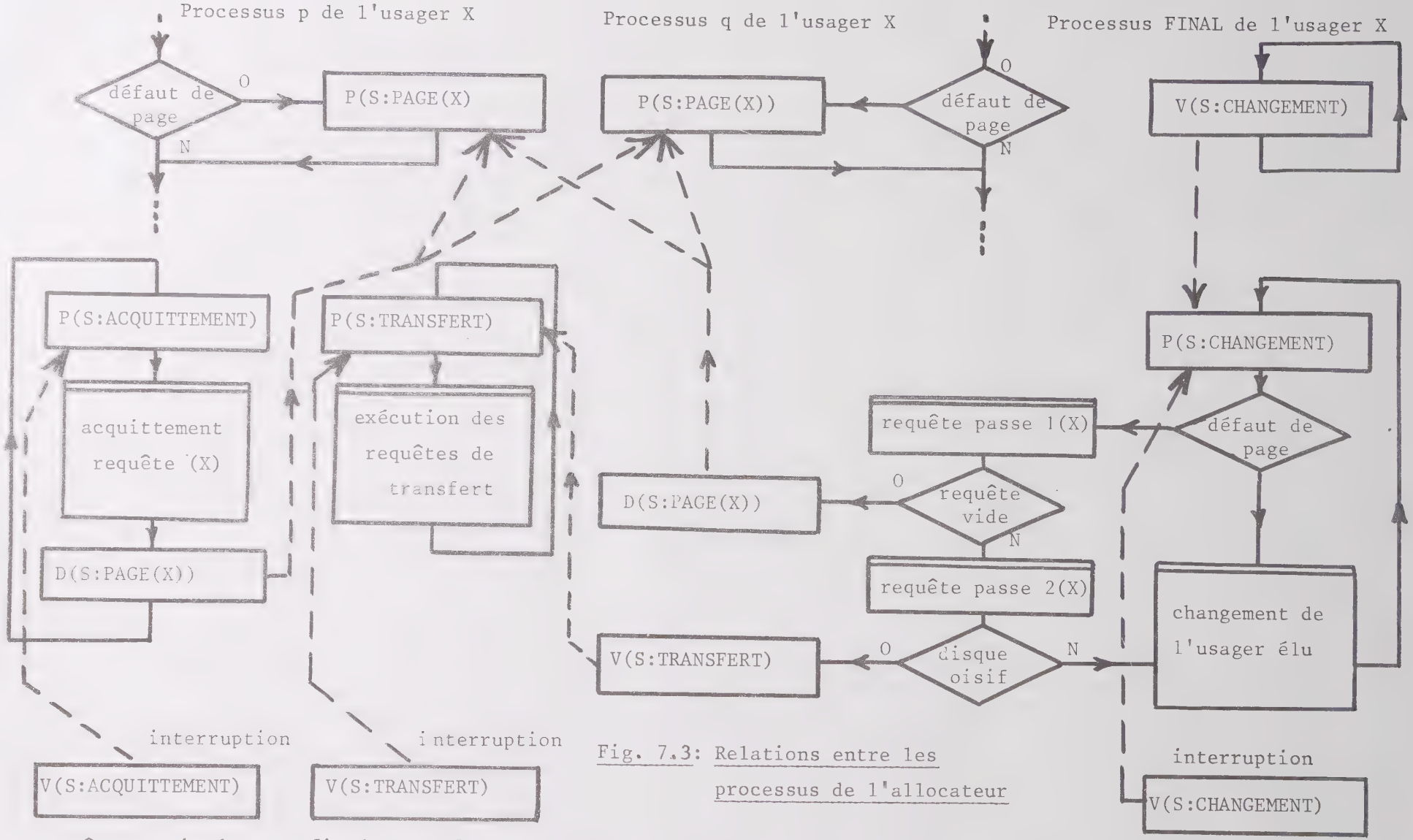


Fig. 7.3: Relations entre les processus de l'allocateur

Les lignes en tirets représentent les relations d'activation.

Le processus d'acquittement des requêtes a la même structure cyclique que les autres processus du moniteur :

```
processus ACQUITTER REQUETE ;
    begin
REPOS : P(S:ACQUITTEMENT) ;
    X:= émetteur de la requête acquittée ;
    if X = 0 then
        acquittement pour le système
    else
        acquittement pour l'usager (X) ;
    goto REPOS
    end
```

Le schéma de la figure 7.3 donne une vue synoptique des relations entre processus, en régime permanent (c'est-à-dire à nombre d'usagers constant). Les opérations d'admission et de départ des usagers vont maintenant être examinées.

7.44 Admission et départ des usagers

Les processus décrits jusqu'ici assurent le fonctionnement en régime permanent du cycle des usagers. Deux processus particuliers sont respectivement chargés de l'introduction d'un nouvel usager dans le cycle et du départ d'un usager.

7.441 Admission

Le processus INITIATEUR, chargé de l'admission d'un nouvel usager, assure les fonctions suivantes :

- allouer à l'usager toutes les ressources qui lui sont initialement nécessaires,
- faire un certain nombre d'initialisations permettant le fonctionnement du nouvel usager,
- introduire le nouvel usager dans le cycle.

L'INITIATEUR est activé :

- lors de l'entrée d'un nouvel usager conversationnel (caractérisée par la réception d'un caractère sur un terminal jusque là inactif),
- lors de l'introduction d'un nouveau travail de fond.

Les ressources à allouer sont :

- une page de garde, destinée à servir de tampon de tampon de communication entre l'usager et le système, et à contenir diverses tables propres à l'usager (TPV, TPD, TFU). Pour éviter de compliquer le processus d'acquittement des requêtes, seules les pages libres sont utilisables (il n'y a donc pas de transfert). Si aucune page n'est libre et disponible (non réservée), l'initiateur se bloque derrière un sémaphore d'attente de page. Une opération D est faite sur ce sémaphore lorsqu'une libération de pages fait passer de 0 à une valeur positive le nombre de pages libres, suivant le schéma ci-dessous :

```
        demande de page
        P(MUTEX) ;
TEST :  if (NPL = 0) & (NPDISP) = 0 then
            begin
            V(MUTEX) ;
            P(SPL) ;
            goto TEST
            end ;
        V(MUTEX) ;
```

```
        libération de page
        P(MUTEX);
        if (NPL = 0) & (NPDISP) = 0 then
            begin
            libération de pages ;
            V(SPL) ;
            end
        else libération de pages ;
        V(MUTEX);
```

La Table des Pages Virtuelles, contenue dans la page de garde, et initialisée, créant ainsi un feuillet pour le nouvel usager.

- Dans le cas d'un usager conversationnel, le terminal émetteur de l'appel lui est alloué pour toute la durée de son existence.

- Un processus premier est créé pour l'usager. Le fichier contenant le programme et les données de ce processus est couplé dans l'espace virtuel de l'usager. Ce processus est créé à l'état actif ; son compteur ordinal

correspondant au début de la procédure d'admission de l'usager (voir plus loin § 7.45).

- Enfin, l'usager est introduit, à l'état oisif, dans le cycle des usagers.(La procédure d'entrée dans l'état oisif est plus simple que celle d'entrée dans l'état prêt). A sa première activation, le processus CHANGEMENT de détectera un processus actif (le processus premier) chez le nouvel usager, qui entrera donc dans la file P par la procédure standard.

Le schéma du processus INITIATEUR est donc :

```
        begin
REPOS : P(S:INITIATEUR) ;
        Allouer un numéro interne d'usager ;
        Allouer une page de garde ;
        Allouer terminal (s'il y a lieu) ;
        Créer processus premier (état actif) ;
        Initialiser le feuillet de l'usager ;
        Introduire l'usager parmi les oisifs ;
        goto REPOS
        end
```

7.442 Départ

Le processus BALAYEUR est chargé de faire sortir un usager du système. Il est activé dans deux cas :

- départ volontaire de l'usager (fin du travail de fond, ou commande 'DEPART' d'un client conversationnel),
- départ forcé par exclusion (l'exclusion d'un usager actif est une opération réservée à l'administration du système).

Les ressources allouées à l'usager par l'INITIATEUR sont libérées.

Le schéma du processus BALAYEUR est :

```
        begin
REPOS : P(S:BALAYEUR) ;
        Libérer la page de garde ;
        Libérer le terminal (s'il y a lieu) ;
        Extraire l'usager de la file où il se trouve ;
        Rendre disponible le numéro de l'usager ;
        goto REPOS
        end
```

7.45 Processus premier

Le processus premier de chaque usager joue pour cet usager le rôle d'un "moniteur décentralisé" ; il assure les fonctions suivantes :

- admission et départ de l'usager
- interprétation du langage de commande (création et destruction des processus traducteurs, gestion de fichiers, entrées/sorties)
- traitement des erreurs, mise au point pour les processus de l'usager
- fonctions diverses d'observation et de mesure
- fonctions d'administration du système (client privilégié ESOPE).

Le processus premier de l'usager "temps réel" assure en outre un certain nombre de fonctions spécifiques liées à la présence de terminaux multiples (allocation de téléimprimeurs, de tampons, communication de messages entre terminaux).

La structure du processus premier est décrite en détail dans [FI] ; nous insisterons surtout ici sur ses relations avec les autres processus du système.

En dehors des phases transitoires où sont traités l'admission de l'usager dans le système et son départ, le processus premier possède, comme les autres processus du moniteur, une structure cyclique avec un état de repos. A la sortie de cet état de repos se situe une phase d'analyse destinée à déterminer la cause de l'activation. Le processus attend ensuite une commande de l'usager. S'il s'agit d'une commande directement exécutable par le processus premier, l'exécution est suivie d'un retour à l'état d'attente de la commande suivante ; sinon, un processus "traducteur" est créé, et le processus premier retourne à l'état de repos.

Les relations entre le processus premier et les processus INITIATEUR et BALAYEUR sont schématisées sur la figure 7.4.

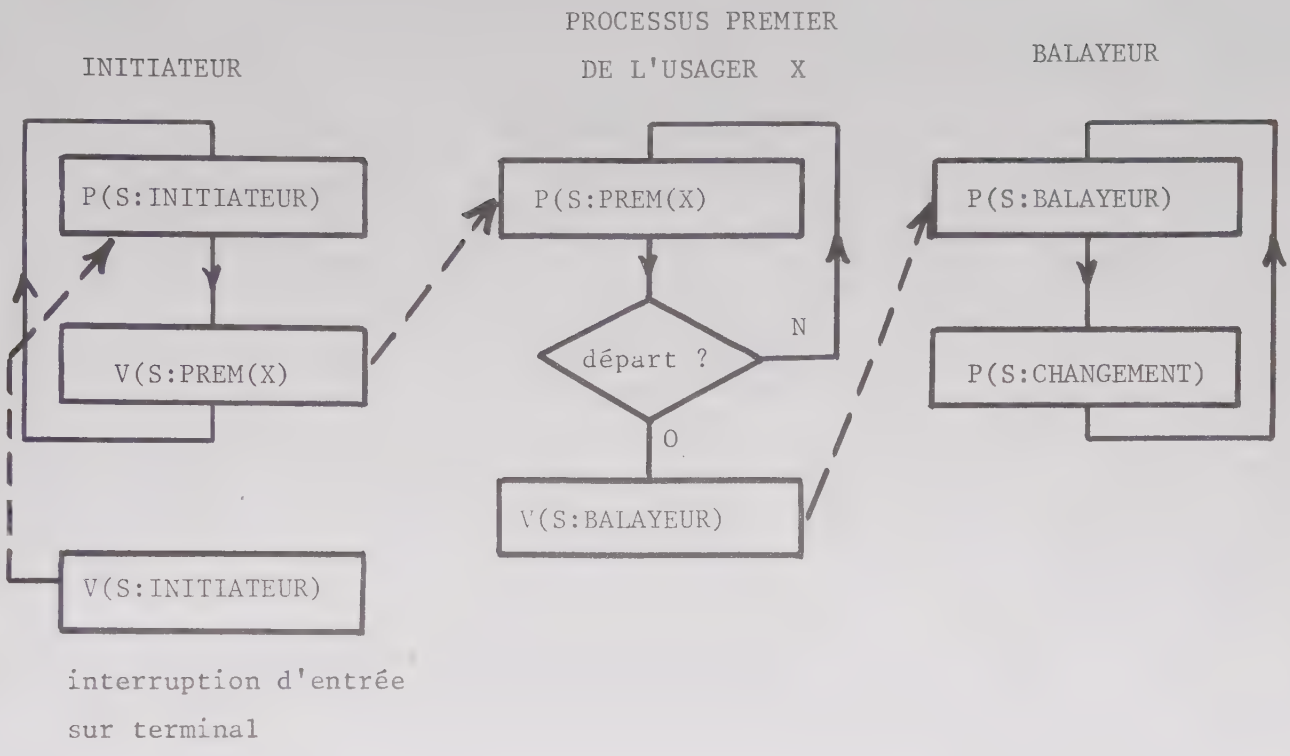


Fig. 7.4 : Admission et départ des usagers

Troisième Partie

Résultats Expérimentaux

Chapitre 8 - Modèle et résultats de simulation du système ESOPE

The purpose of computation
is insight, not numbers

R.W. Hamming

Chapitre 8

MODELE ET RESULTATS DE SIMULATION DU SYSTEME ESOPE

8.1 Objectifs de la simulation

Le modèle de simulation du système ESOPE a été construit parallèlement au système lui-même ; son objectif principal était d'étudier les stratégies fondamentales du système avant leur mise en oeuvre effective.

Le but de ces études est donc moins de prévoir des performances en valeur absolue que de tester la validité de certaines hypothèses, de confirmer (ou infirmer) des conclusions fondées sur la seule intuition, de pouvoir évaluer quantitativement, au moins en valeur relative, les résultats de telle décision de conception.

L'expérimentation a été centrée sur les problèmes de gestion de l'ensemble processeur-mémoire centrale-mémoire secondaire. Il nous a semblé, pour les raisons exposées au chapitre 1, que les problèmes d'entrée-sortie pouvaient être laissés de côté pour cette étude spécifique. De même, a été éliminé du modèle, pour des raisons de simplicité, le travail de fond, qui représente essentiellement une charge de l'unité centrale pendant les périodes laissées libres par les autres travaux, ainsi qu'un ralentissement moyen du trafic de pages des autres travaux. Le système simulé comprend donc un ensemble d'usagers conversationnels, le comportement des différents usagers obéissant à la même loi statistique. (cf § 8.23).

Par ailleurs, on ne s'est intéressé qu'au comportement du système en régime permanent, c'est-à-dire à nombre d'usagers constant. Il a été jugé que la simulation de l'arrivée et du départ d'usagers compliquerait le modèle sans apporter d'informations utiles. Une analyse des résultats de simulation est présentée dans (B5).

8.2 Modèle de simulation

8.21 Caractéristiques générales

Le simulateur peut se décomposer en trois parties logiquement distinctes :

- le moniteur, chargé de gérer les usagers et les ressources principales : mémoire centrale, mémoire secondaire, et unité centrale,
- la charge du système, constituée par un nombre réglable d'usagers, travaillant en mode conversationnel à partir de terminaux,
- le programme de contrôle, chargé d'initialiser les simulations et de recueillir les statistiques.

Dans la construction de tout modèle, il y a conflit entre la simplicité de réalisation et la fidélité au système représenté. On a essayé de représenter le plus fidèlement possible les mécanismes centraux d'allocation de ressources ; d'autres parties du système ont reçu un traitement plus global : ainsi il n'a pas été jugé utile, lors de la simulation des interactions avec les terminaux, de simuler l'entrée individuelle des caractères ; on s'est contenté de définir un comportement statistique de la longueur des messages, et d'imputer à l'unité centrale le temps de traitement correspondant. Plus généralement, une méthode simplificatrice a été utilisée à plusieurs reprises : elle consiste à isoler un sous-système ayant des relations bien définies avec le reste, à étudier expérimentalement son comportement moyen, et à remplacer dans le modèle général le sous-système par un modèle traduisant ce comportement. Cette méthode permet de réduire de façon notable le temps d'exécution de la simulation.

Le simulateur a été écrit dans le langage SIMULA 1.

8.22 Simulation du moniteur

La gestion de la mémoire et de l'unité centrale sont fidèlement reproduites. Un processus (dans ce paragraphe, ce mot sera employé au sens du langage SIMULA) est chargé d'admettre les usagers en mémoire et de leur allouer l'unité centrale. L'accès en mémoire est régi par un test d'admission (cf § 6.4).

Le fonctionnement des disques à têtes fixes est simulé par un processus activé lors du lancement d'une requête ; le temps de transfert des pages est un facteur essentiel du temps de réponse des usagers et doit donc être déterminé avec précision. L'optimisation de l'exécution des requêtes, par modification de

l'ordre des transferts, est simulée sans avoir à tenir compte des adresses individuelles des pages sur disque. On utilise pour cela les courbes données au chapitre 6 (fig. 6.9), déterminées par une simulation préliminaire, et représentées approximativement par une formule algébrique.

On obtient ainsi une évaluation du temps moyen de transfert d'un groupe de pages en fonction du nombre de pages transférées. Les adresses de début de pages sont supposées réparties uniformément sur les disques, hypothèse justifiée compte tenu du grand nombre de transferts.

8.23 Simulation de la charge du système

La charge du système est constituée d'un nombre variable de terminaux en interaction directe. Le comportement des usagers est essentiellement cyclique (cf § 2.2). Le temps de réflexion est schématisé par une variable aléatoire de distribution uniforme, variant entre 2 et 20 secondes, ce temps comprenant l'entrée et la sortie de messages sur le terminal. Le temps de réponse d'une interaction est défini comme l'intervalle de temps compris entre la fin de l'entrée d'une demande et le début de la sortie de la réponse suivante. L'exécution d'un programme se représente par un processus générateur de tirages aléatoires de numéros de pages virtuelles ; si la page est reconnue manquante, un déroutement est simulé pour demander le chargement de cette page. Le temps de traitement moyen d'une interaction a été fixé à 30 ms.

Etant donné la mauvaise connaissance du comportement réel des programmes, on a choisi la modèle aussi simple que possible, avec toutefois la possibilité de simuler un comportement "local", et d'agir sur les paramètres de ce comportement. Le modèle choisi est le modèle à probabilité de référence constante, avec "noyau", décrit au § 2.321. Ce modèle présente l'avantage de pouvoir mettre aisément en évidence les phénomènes dépendant de la localité du comportement. Toutefois, en raison même de sa simplicité, il ne saurait fournir une représentation fidèle d'une charge réelle. En conséquence, les résultats obtenus à l'aide du simulateur ne doivent pas être pris comme des prédictions absolues de performance, mais doivent être utilisés de manière comparative, par exemple pour évaluer l'efficacité relative de plusieurs stratégies ou pour apprécier l'influence de la variation de paramètres physiques (taille de mémoire, vitesse des disques).

8.3 Exploitation du modèle

8.31 Remarques générales

Avant de réaliser les expériences proprement dites, il est nécessaire

de vérifier la validité du simulateur, d'évaluer la précision de ses résultats, et de figer un certain nombre de paramètres ; il n'est pas possible en effet, ne serait-ce qu'en raison du temps élevé de la simulation, de faire évoluer tous les paramètres. Parmi les nombreux paramètres du modèle, un certain nombre peuvent être fixés une fois pour toutes, si l'on estime que la variation de ces paramètres dans un certain domaine n'a pas d'influence sur les résultats, ou a une influence entièrement prévisible. C'est ainsi que le temps moyen de changement d'usager a été fixé à 3 ms et la durée moyenne d'une interaction à 30 ms.

Tous les résultats obtenus sont de nature statistique, et il est donc important d'avoir une idée de leur précision et de leur stabilité. Une expérience de simulation comprend l'échange d'un certain nombre de messages par terminal, nombre qui détermine la durée de l'expérience. Le tableau 8.1 montre l'influence de ce paramètre sur le temps de réponse moyen et l'écart-type mesuré, les autres paramètres étant maintenus constants.

Nombre de messages	Temps de réponse moyen (s.)	Ecart-type
40	1.81	1.01
100	1.87	1.12
200	1.78	1.07
400	1.78	1.08
800	1.90	1.05

Tableau 8.1

Pour les expériences dont il est rendu compte ici, la durée moyenne des essais a été fixée à 200 messages, ce qui semble correspondre à une erreur de 5 à 10 % environ sur le temps de réponse moyen.

8.32 Utilisation du modèle

Une expérience consiste à exécuter une série de simulations en faisant varier un et parfois deux paramètres, et en modifiant une stratégie donnée du moniteur. Les paramètres fondamentaux sont : le nombre d'usagers en activité, la taille de la mémoire allouable, les paramètres définissant le comportement des

programmes....).

Comme il vient d'être indiqué, une simulation se termine lorsque 200 messages ont été échangés (ce qui correspond à environ 3 minutes de fonctionnement du système réel sous une charge de 20 usagers). Notons que toutes les expériences se déroulent avec un système en régime permanent, c'est-à-dire avec un nombre constant d'usagers.

Les principales variables mesurées sont les suivantes :

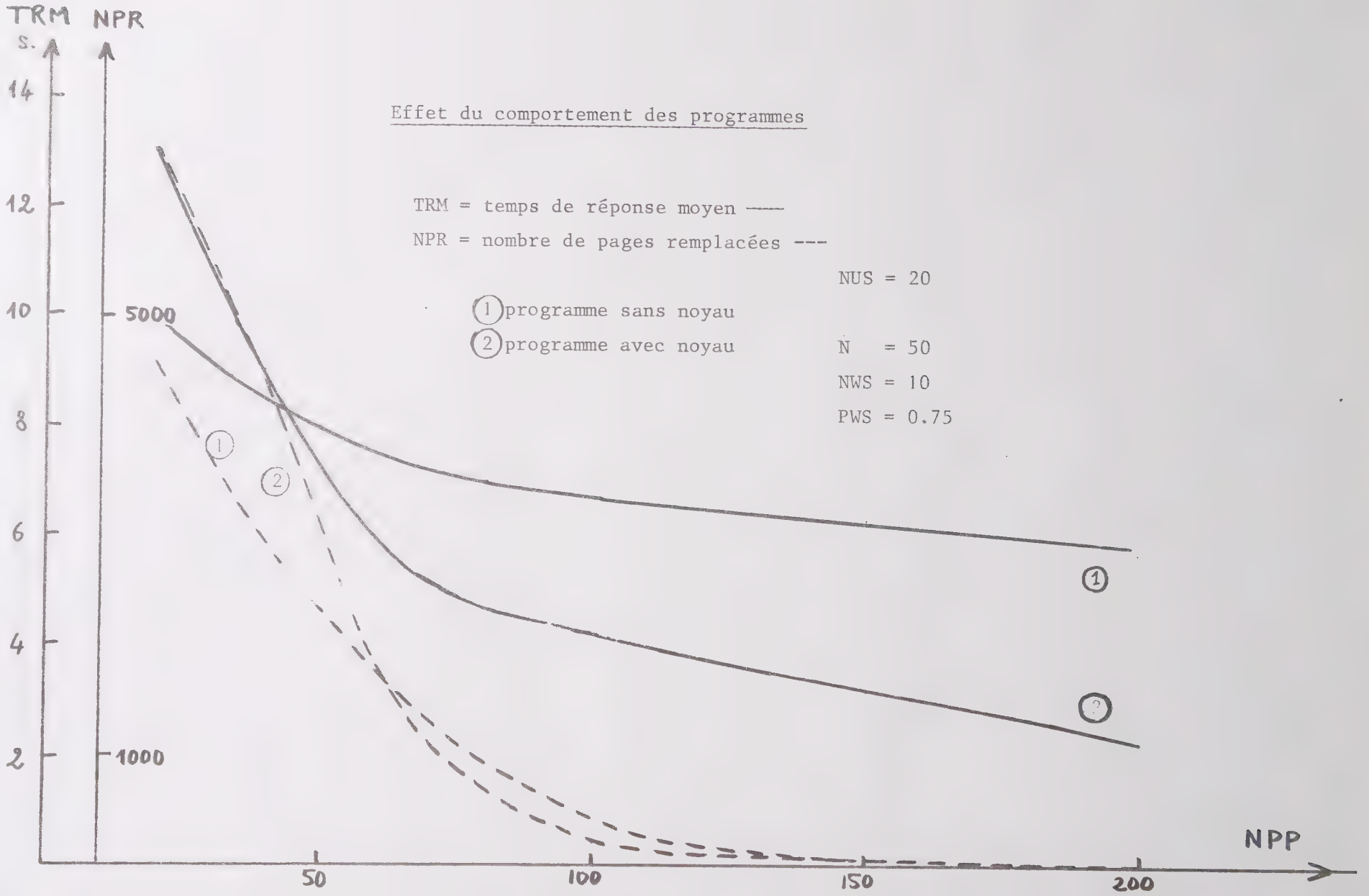
- répartition de l'activité de l'unité centrale : en attente d'usagers, en attente de page, exécution de programmes du moniteur, exécution de programmes d'usagers.
- activité des disques : facteur d'utilisation, nombre de pages transférées dans les deux sens, taille moyenne des requêtes.
- statistiques sur les pages physiques : pages chargées, vidées, récupérées, partagées.
- temps de réponse (moyenne, écart-type).
- longueur moyenne des files d'usagers prêts et courants.

8.4 Gestion de la mémoire physique

Une première série d'expériences a été consacrée à la politique de gestion de la mémoire. Cette étude a été centrée sur la mise en évidence, l'explication et la prévention du phénomène d'écroulement (cf. chapitre 3). Rappelons que ce phénomène, qui se manifeste sur les systèmes paginés utilisant la politique de la page à la demande, se manifeste par une chute brutale des performances du système, liée à un accroissement des échanges de pages et à un taux d'oisiveté élevé de l'unité centrale. On a vu que le phénomène d'écroulement est lié aux caractéristiques de comportement de programmes et à la stratégie d'allocation de la mémoire : on étudiera donc l'influence de ces deux facteurs.

8.41 Influence du comportement des programmes

Une première série d'expériences a été faite en supprimant la réservation de mémoire à l'entrée des usagers dans l'état courant. L'état "prêt" disparaît : un usager débloqué passe aussitôt à l'état courant. En conséquence, un défaut de page peut se produire alors qu'il n'y a pas de page libre : on remplace dans ce cas au hasard une page déjà allouée. Pour mettre en évidence l'influence du comportement local, on a comparé un modèle de programmes à références uniformes avec un modèle comportant un noyau. La figure 8.1 représente, pour une même charge de 20 usagers, l'évolution du temps de



Effet du comportement des programmes

TRM = temps de réponse moyen ——

NPR = nombre de pages remplacées ---

NUS = 20

(1) programme sans noyau

(2) programme avec noyau

N = 50

NWS = 10

PWS = 0.75

Figure 2.1

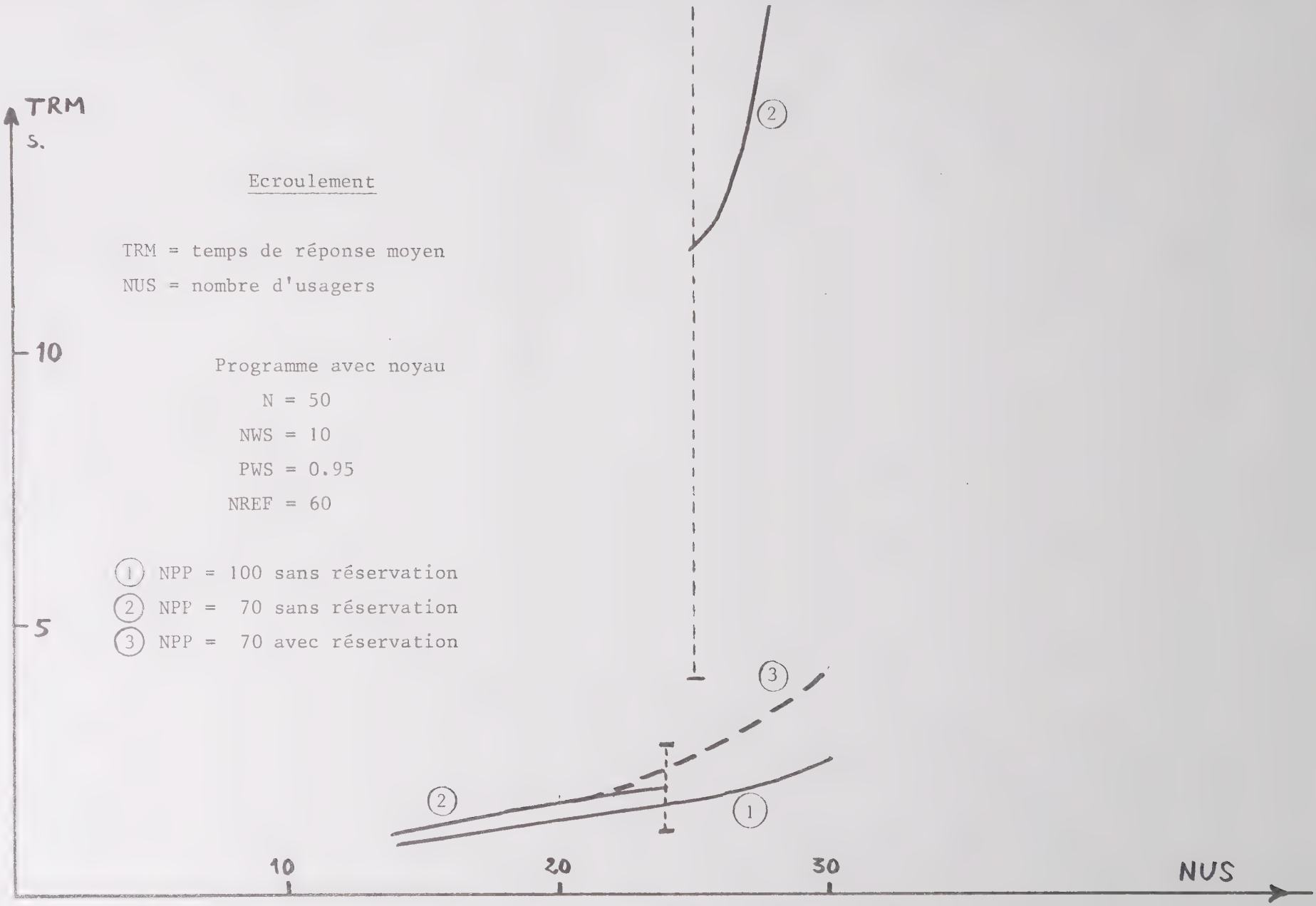

TRM
S.
Ecroulement
TRM = temps de réponse moyen
NUS = nombre d'usagers
10
Programme avec noyau
N = 50
NWS = 10
PWS = 0.95
NREF = 60
1 NPP = 100 sans réservation
2 NPP = 70 sans réservation
3 NPP = 70 avec réservation
5
2
3
2
1
10
20
30
NUS

Figure 8.2

réponse moyen avec le nombre de pages physiques allouables. On constate que les performances du programme avec noyau deviennent rapidement meilleures lorsque la taille de la mémoire augmente.

Dans un programme avec noyau, les pages du noyau sont référencées plusieurs fois, d'où un gain en nombre de transferts de page par rapport au programme à références uniformes ; mais à mesure que la taille de la mémoire diminue, la probabilité de pouvoir réutiliser une page de noyau sans avoir à la recharger décroît, si bien que le nombre de défauts de page tend, pour les deux comportements de programme, à rejoindre le nombre de tirages de page.

Ce phénomène est également illustré sur la figure 8.1, où on a représenté le nombre de remplacements de pages.

En conséquence, en l'absence de réservation de mémoire, on voit que les programmes à comportement local sont favorisés lorsqu'il y a peu de remplacement, c'est-à-dire quand la taille de mémoire disponible est grande ; en revanche, en mémoire restreinte, la dégradation des performances de ces programmes est plus rapide.

8.42 Influence de la politique de réservation de mémoire

On s'intéresse maintenant au comportement du système en fonction de la charge. En l'absence de mécanisme limitant le degré de multiprogrammation, l'écroulement est prévisible (cf chapitre 3) lorsque la taille de mémoire disponible est insuffisante pour contenir l'ensemble des "working sets" des usagers courants.

Ce phénomène d'écroulement est clairement visible sur la figure 8.2 représentant le temps de réponse moyen en fonction de la charge. Pour une taille de mémoire allouable de 70 pages, la charge étant constituée de programmes comportant un noyau de 10 pages, on constate une discontinuité du temps de réponse lorsque la charge dépasse 25 usagers. L'écroulement se manifeste en outre par :

- une dispersion beaucoup plus grande du temps de réponse autour de sa valeur moyenne (voir fig. 8.2)
- une croissance très forte du nombre de remplacements de pages.

Le paramètre $\frac{NPP}{NWS}$ représente le nombre moyen d'usagers courants au-dessus duquel l'écroulement se produit ; dans les conditions de l'expérience, ce paramètre vaut 7. On constate en effet que le taux d'occurence d'un nombre d'usagers courants supérieur ou égal à 7 croît brutalement au-dessus de 25 usagers. Le tableau 8.2 représente la variation des principaux paramètres énumérés :

NUS	Nb observations ⩾7 usagers courants	Nb remplacements	Nb échanges de pages	Ecart-type temps de réponse
22	6	13	3520	0,9
24	4	9	3600	0,8
25	135	3930	8180	8,4
26	122	3900	8160	9,5
28	153	5230	9740	9,5

Tableau 8.2

On constate (fig.8.2) que l'introduction de la politique de réservation de mémoire permet de supprimer l'écroulement à 25 usagers. (pour des raisons techniques, la simulation n'a pu être poursuivie au-delà de 30 usagers). Par ailleurs, on a également représenté sur la figure 8.2 la courbe de temps de réponse obtenue pour une taille de mémoire de 100 pages, sans réservation. Le seuil d'écroulement est reporté au-delà de 30 usagers et n'a pu être observé.

8.5 Etude des échanges de pages

Un objectif important de la politique d'allocation de ressources est la réduction des échanges de pages entre mémoire et disque. On a déjà examiné au chapitre 6 la réduction du temps d'exécution des requêtes. Un second mode d'optimisation consiste à anticiper sur les transferts prévisibles ou probables. Suivant le sens du transfert, on parlera de préchargement ou de prévidage.

8.51 Préchargement

Le principe du préchargement consiste à amener en mémoire centrale, lors du passage d'un usager à l'état courant, des pages dont la probabilité estimée d'utilisation par cet usager est élevée. Les avantages attendus du préchargement sont les suivants :

- réduction du temps consacré à la gestion du système (grâce à la diminution du nombre de défauts de page,
- optimisation du transfert par allongement de la taille moyenne des requêtes, à nombre de pages transférées constant.

Cette technique implique toutefois le risque de charger des pages qui ne seront pas utilisées ; l'intérêt du préchargement est donc directement lié à l'évaluation de la probabilité d'utilisation des pages.

L'expérience faite en simulation a consisté à évaluer les avantages du préchargement dans le cas où cette probabilité d'utilisation est connue. Le modèle de programme utilisé comportant un noyau, seules les pages de ce noyau ont été préchargées. Les résultats de l'expérience, avec et sans préchargement, sont résumés dans le tableau 8.3 ; on s'est d'abord placé dans le cas où le noyau est "dur", cas où le préchargement des pages du noyau doit donner des résultats favorables.

	Nombre d'usagers	sans préchargement	avec préchargement
Taille moyenne des requêtes (pages)	10	1.5	2.5
	20	2.9	5.5
	30	5.0	9.0
Nombre de requêtes	10	2470	1470
	20	1710	910
	30	960	560
Taux d'oisiveté des disques	10	0.55	0.63
	20	0.16	0.33
	30	0.07	0.15
Fraction de temps passé à la gestion du système	10	0.07	0.04
	20	0.13	0.06
	30	0.16	0.07
Fraction de l'unité centrale disponible	10	0.91	0.94
	20	0.84	0.91
	30	0.79	0.88
Temps de réponse moyen (s.)	10	1.10	0.98
	20	3.19	2.49
	30	8.41	7.33

Tableau 8.3

Effet du préchargement

Paramètres : NPP = 100

Programme avec noyau { N = 50, NWS = 10, PWS = 0.80

Pas de pages partagées

Ces résultats donnent lieu aux remarques suivantes :

- avec le préchargement, la taille moyenne des requêtes est augmentée, et leur nombre diminué, le nombre total de pages transférées restant sensiblement constant (il est légèrement supérieur avec préchargement, en raison de l'incertitude sur l'utilisation des pages préchargées). L'allongement des requêtes a pour effet d'augmenter le taux d'oisiveté des disques.

- la fraction du temps de processeur consacrée à la gestion du système est réduite, ce qui semble devoir être attribué au nombre moindre de défauts de page ; le taux d'oisiveté de l'unité centrale est corrélativement augmenté.

- les facteurs précédents donnent lieu à une légère amélioration du temps de réponse moyen.

En fait, dans les conditions habituelles d'utilisation d'un système d'exploitation, il est souvent difficile d'estimer a priori avec précision la probabilité d'utilisation des pages. Il était donc intéressant d'examiner l'influence comparée du préchargement pour plusieurs types de comportement. On a fait varier la taille du noyau entre 0.33 et 1 fois la taille du programme ; le dernier cas correspond à une ignorance complète du comportement du programme. Les résultats sont résumés, en ce qui concerne le temps de réponse moyen, dans le tableau 8.4.

Taille relative du noyau	temps de réponse moyen	
	sans préchargement	avec préchargement
1/3	2.21	1.65
1/2	5.42	5.09
1	9.80	10.5

Tableau 8.4

On constate que l'influence favorable du préchargement diminue, et tend même à s'inverser lorsque la qualité de la prévision des pages à précharger diminue.

4.33 Prévidage

Le principe du prévidage consiste à utiliser le temps d'oisiveté du canal de communication entre mémoire centrale et disques pour transférer vers ceux-ci des pages quasi-libres. Le but recherché est d'augmenter le nombre de pages libres directement réutilisables, et donc de réduire le temps moyen d'allocation des pages. Un risque possible est que l'exécution des requêtes de prévidage retarde la prise en compte de requêtes pour le compte des usagers.

On a d'abord examiné l'influence d'une limitation du nombre de pages prévidées. On obtient des courbes ayant l'allure de celle représentée sur la figure 8.3 ; on voit que l'on n'a pas intérêt à limiter le nombre de pages prévidées.

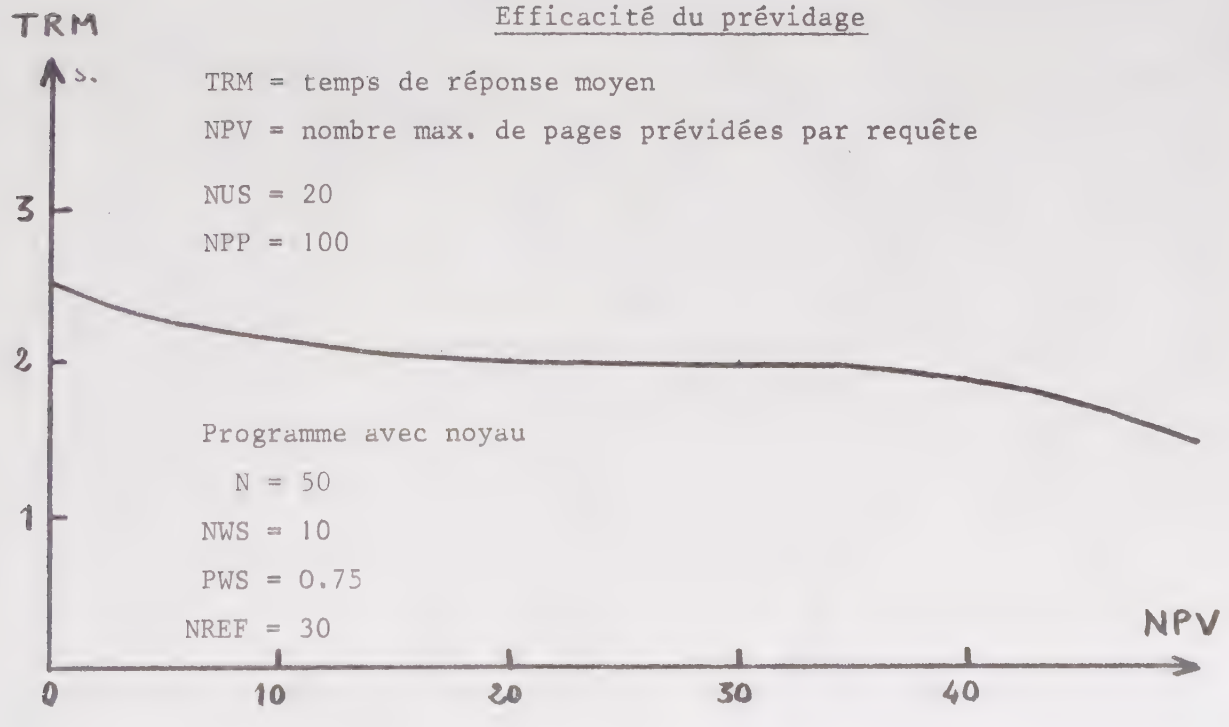


figure 8.3

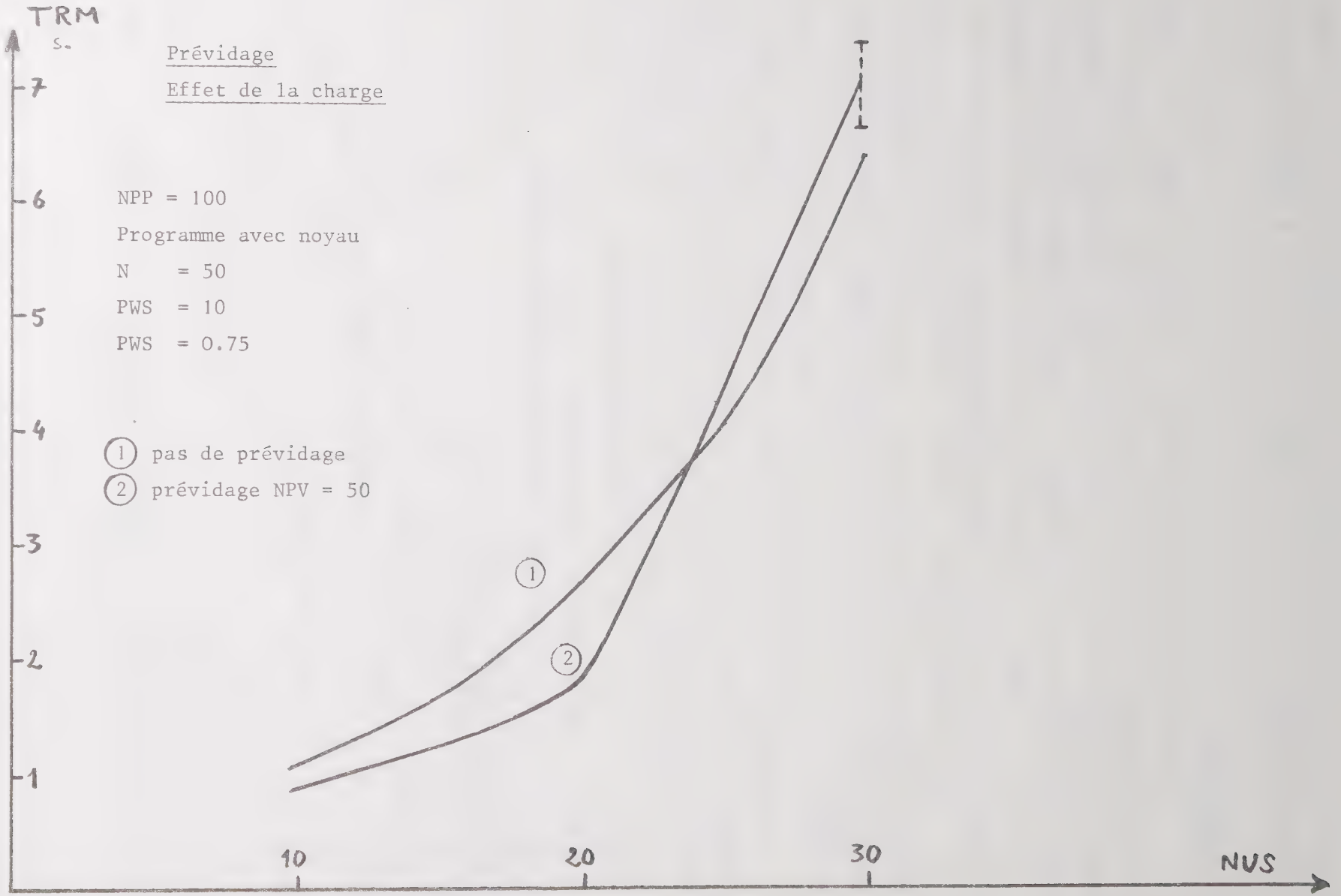

TRM
S.
Prévidage
Effet de la charge
NPP = 100
Programme avec noyau
N = 50
PWS = 10
PWS = 0.75
1 pas de prévidage
2 prévidage NPV = 50
7
6
5
4
3
2
1
10
20
30
NVS

Figure 8.4

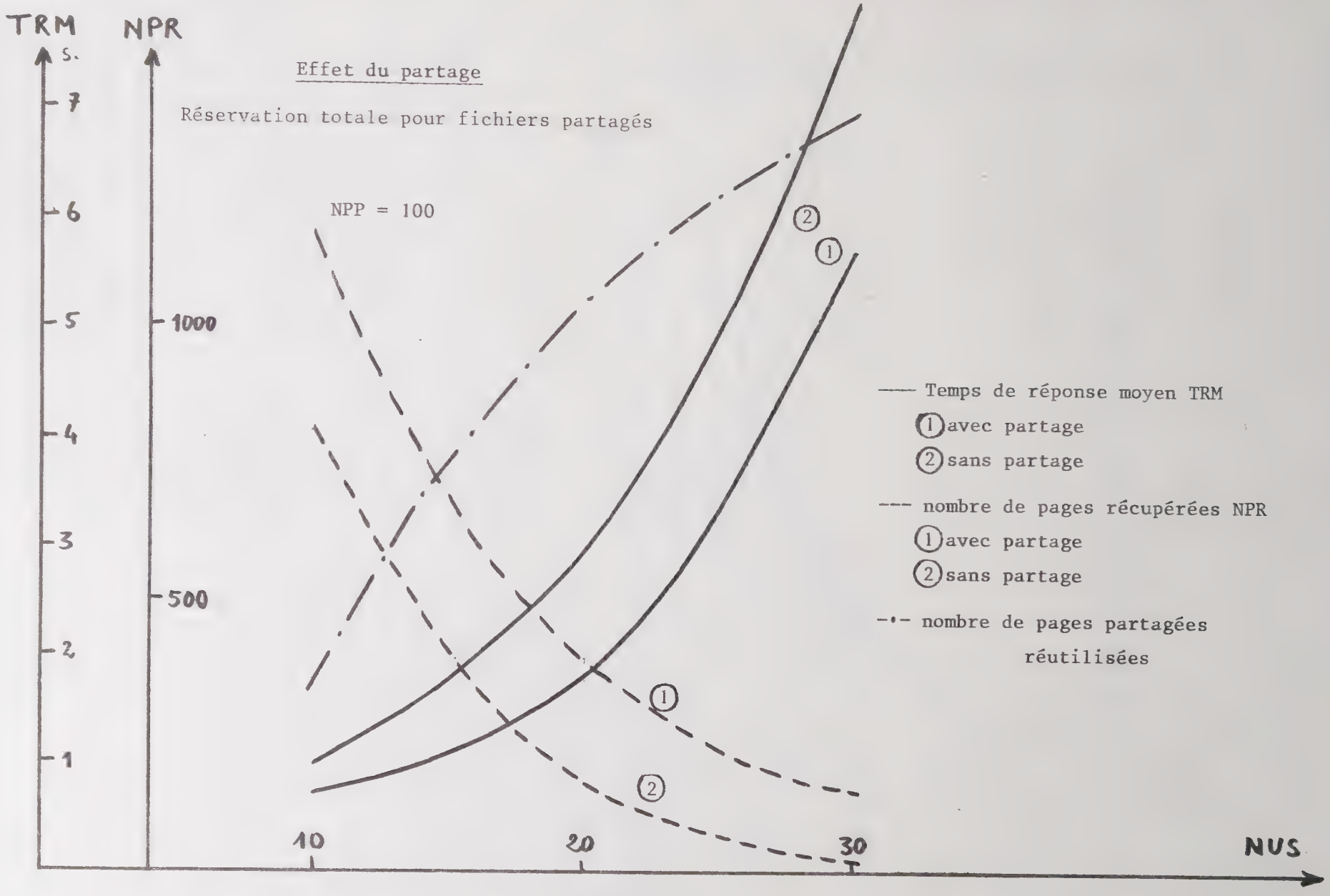

TRM
NPR
s.
Effet du partage
Réservation totale pour fichiers partagés
NPP = 100
7
6
5
4
3
2
1
1000
500
10
20
30
NUS
①
②
Temps de réponse moyen TRM
①avec partage
②sans partage
nombre de pages récupérées NPR
①avec partage
②sans partage
nombre de pages partagées réutilisées

Figure 8.5

On a représenté sur la figure 8.4 la variation du temps de réponse moyen en fonction de la charge avec et sans prévidage. On constate que l'efficacité du prévidage est maximale à charge moyenne ; à charge faible, l'avantage est négligeable ; à charge élevée, le prévidage tend à avoir un effet défavorable sur le temps de réponse.

Expliquons ce comportement : à faible charge, la réserve de pages libres suffit en grande partie à satisfaire les demandes de pages ; son extension par le prévidage ne procure donc qu'un avantage réduit. A charge très élevée, l'activité du canal des disques est grande, et la probabilité d'interférence des requêtes de prévidage avec les autres requêtes augmente. Cela entraîne un retard moyen qui se traduit par l'accroissement du temps de réponse.

8.6 Partage de l'information

Dans le système ESOPE, les programmes communs à plusieurs usagers sont écrits de façon réentrante, ce qui permet de partager les pages contenant leur code et leurs données invariantes. Ce partage réduit le nombre de chargements de pages, et la place occupée en mémoire. On étudie ici les problèmes de gestion des pages partagées, et les actions qui tendent, au niveau de la gestion des usagers, à créer des conditions favorables au partage.

8.61 Gestion des pages partagées

La figure 8.5 montre l'effet global du partage sur le temps de réponse, effet net mais non spectaculaire ; il faut de plus noter que la présence d'un noyau unique dans le programme crée des conditions particulièrement favorables au partage.

Ces conditions seraient moins favorables avec un modèle à localité variable (cf § 2.323) représentant plus fidèlement la réalité.

On également porté sur la figure 8.5 la variation du nombre de pages récupérées, avec et sans partage ; comme on peut s'y attendre, le partage d'information augmente le nombre de pages récupérées ; on peut également noter la croissance, avec la charge, du nombre de pages partagées effectivement réutilisées.

Le problème de la réservation de mémoire pour les fichiers partagés a été abordé au § 6.41. Doit-on réserver pour un fichier partagé un nombre de pages égal à sa taille totale ? La figure 8.6 représente les résultats d'une expérience où on a réservé soit la taille totale soit la taille du noyau avec possibilité de débordement. On constate, dans les conditions de cette

expérience, que la différence de performances est faible.

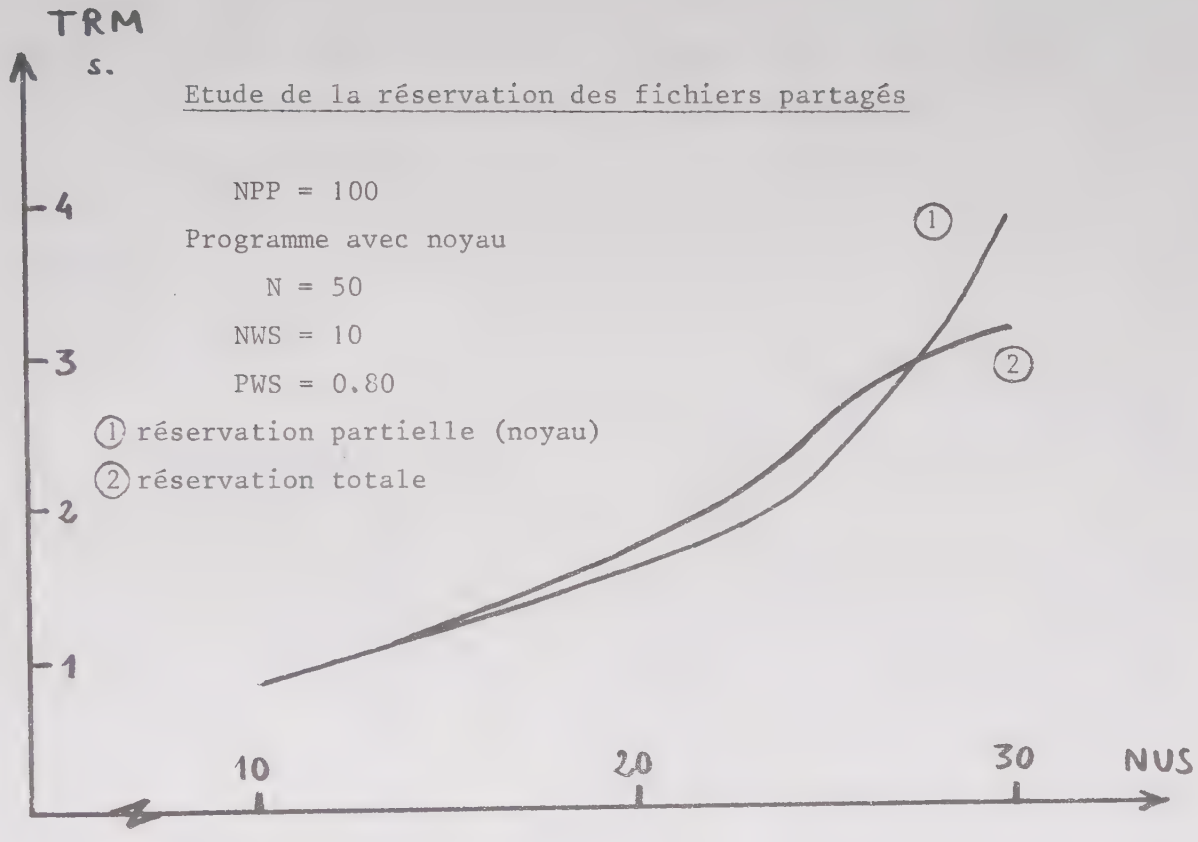


figure 8.6

8.62 Regroupement des usagers

Le partage de pages augmente d'autant plus l'efficacité du système que les usagers utilisant simultanément les informations partagées sont plus nombreux. Aussi est-il naturel de chercher à contrôler le cycle des usagers et la gestion des fichiers partagés de façon à créer des conditions favorables au partage.

L'idée du regroupement des usagers prêts consiste à rassembler, dans la file P, ceux des usagers partageant un même fichier. On augmente ainsi

les chances de ces usagers de se retrouver ensemble dans l'état courant. En fait, l'expérience montre que l'effet du regroupement ne se fait sentir qu'à charge élevée. Cet effet est évidemment fonction de la fraction de la population totale participant au partage. Les résultats représentés sur la figure 8.7 montrent que, dans les conditions de l'expérience, l'effet global du regroupement est nul ou même négatif : le temps de réponse est amélioré pour les usagers participant au partage, mais détérioré pour les autres.

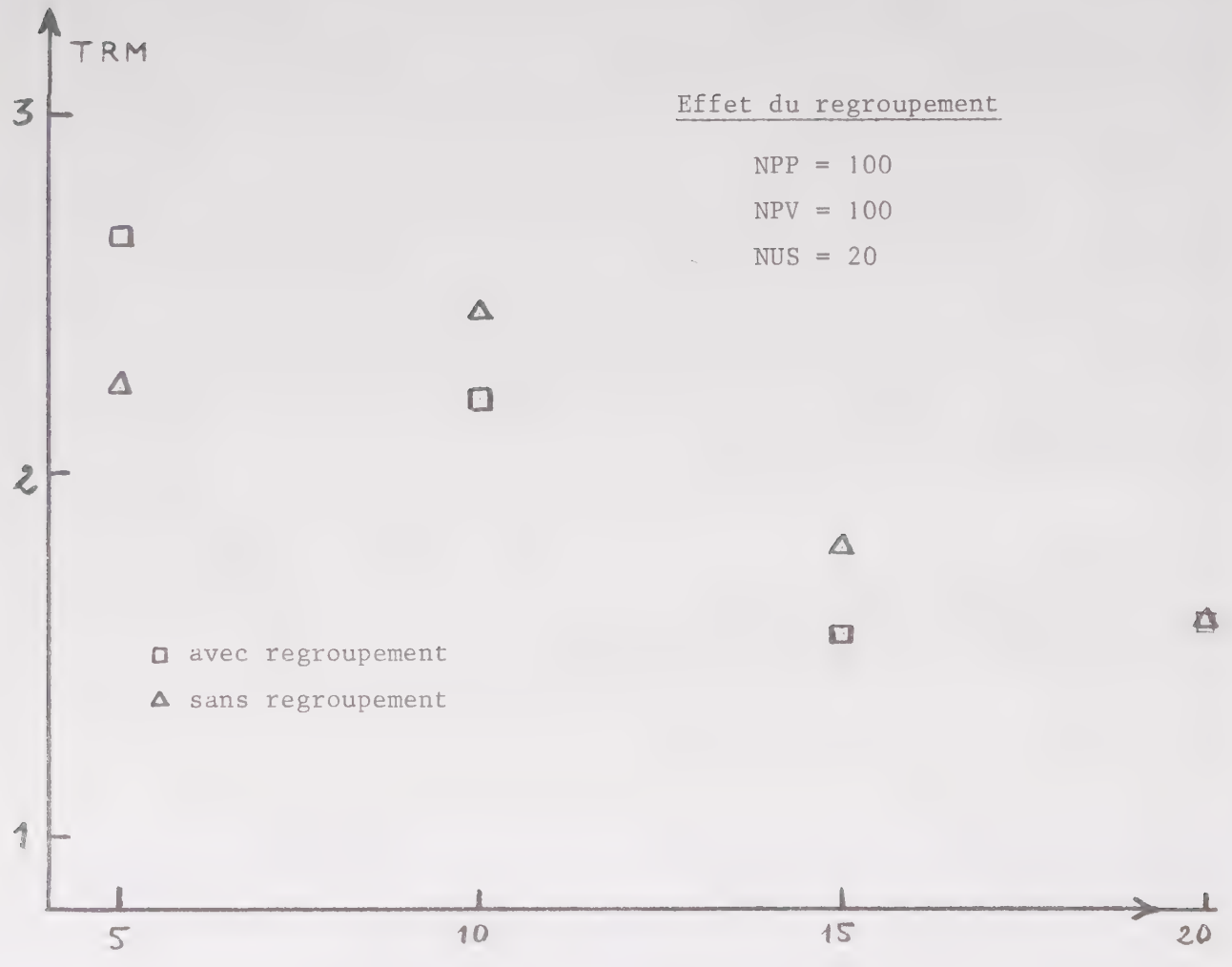


figure 8.7

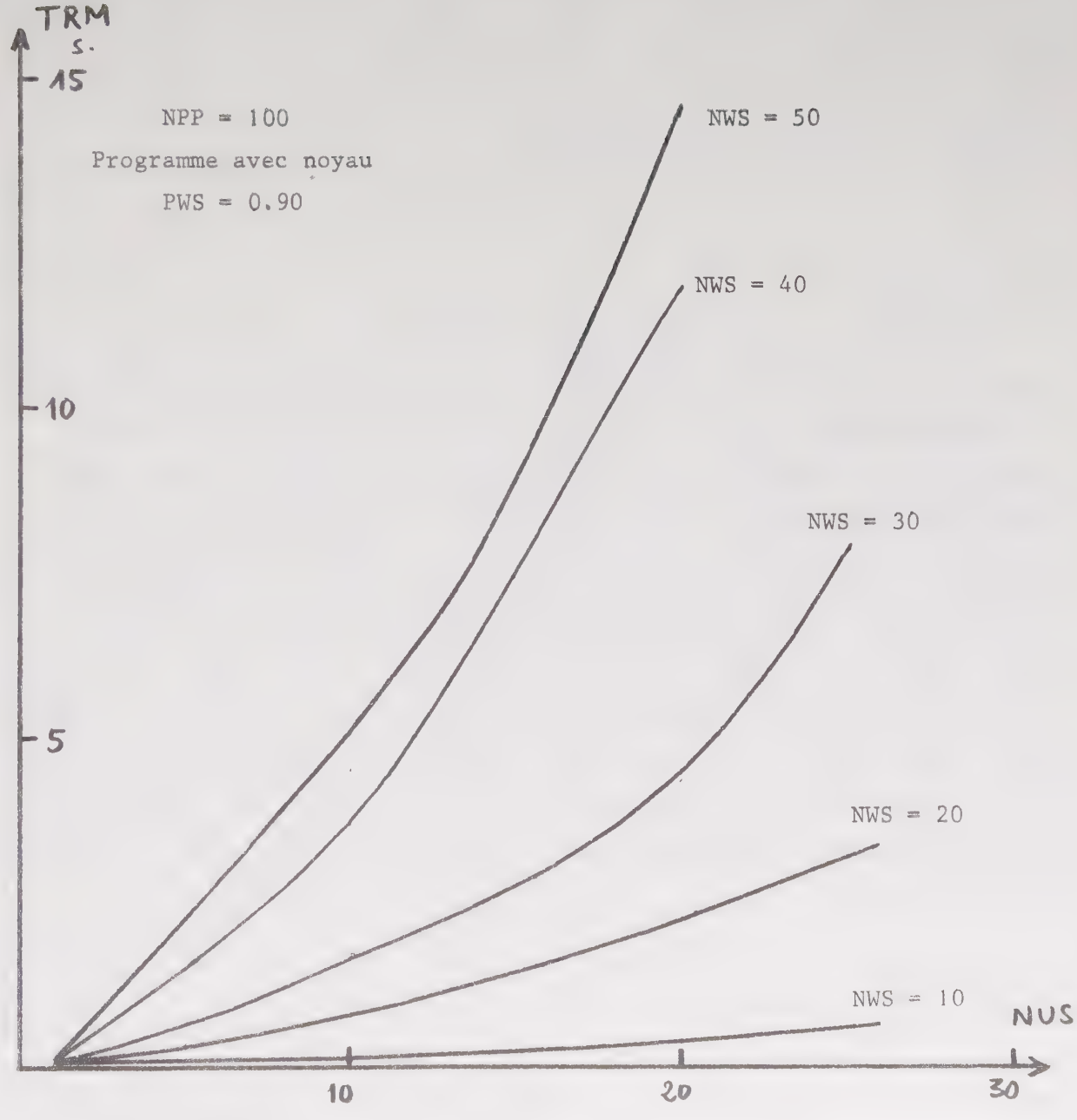

TRM
s.
15
10
5
NPP = 100
Programme avec noyau
PWS = 0.90
NWS = 50
NWS = 40
NWS = 30
NWS = 20
NWS = 10
NUS
10
20
30

figure 8.8

Une autre façon de favoriser le partage d'information est de maintenir les fichiers partagés en mémoire aussi longtemps que possible ; ou plus exactement, tant qu'ils ont une certaine probabilité d'être utilisés. La difficulté réside dans l'estimation de cette probabilité. On utilisé dans ESOPE (cf chapitre 6), un compteur indiquant le nombre d'usagers courants utilisant chaque fichier partagé ; ce fichier reste en mémoire tant que ce compteur est positif.

Une expérience a consisté à faire enregistrer par ce compteur le nombre d'usagers courants ou prêts utilisant le fichier. On constate que cette modification n'améliore pas le temps de réponse de façon visible, mais au contraire défavorise les usagers qui ne participent pas au partage.

Tenant compte du fait que le modèle à noyau unique tend plutôt à favoriser le partage, on peut conclure en définitive au peu d'intérêt des modifications de la stratégie d'allocation tendant à favoriser le partage.

8.7 Conclusion

Cette série de simulations a permis de mettre en évidence le bien-fondé de plusieurs des stratégies d'allocation de ressources utilisées dans le système ESOPE. L'étude sur un modèle permet de dépasser l'intuition qualitative et de se faire une idée, au moins approchée, du comportement quantitatif d'un système. C'est ainsi qu'on a pu constater l'influence bénéfique, sous certaines conditions, de la politique de réservation de mémoire, de l'optimisation du transfert mémoire-disque, du partage des pages. Dans de nombreux cas, en revanche, la comparaison de deux politiques ne permet pas de conclure nettement, les résultats étant équivalents dans l'environnement étudié. Par ailleurs, la simulation permet de détecter les "fausses améliorations" : une modification locale d'une stratégie peut qualitativement apparaître comme un progrès ; elle ne sera en fait bénéfique qu'à la double condition de servir suffisamment souvent, et de ne pas entraîner par ailleurs de conséquences négatives imprévues.

Conclusion

Aux ouvrages que l'on fait, il ne faut pas demander plus ... que de vous apprendre quelque chose.

Paul Valéry
Mauvaises Pensées.

Conclusion

Au terme de cette étude, il nous reste à tirer les enseignements de l'expérience qu'a constitué pour nous la conception et la réalisation du système ESOPE. Nous tenterons dans la mesure du possible de nous dégager des circonstances particulières à ce système pour aboutir à des conclusions d'une portée plus générale ; nous essaierons enfin de placer quelques jalons pour des recherches futures.

Nous adopterons pour cette discussion le plan suivant :

- Rappel des principales idées directrices du projet
- Examen critique des solutions apportées aux différentes classes de problèmes abordés :
  - gestion des processus parallèles
  - accès à l'information
  - notion d'usager, et sa réalisation
  - allocation des ressources
- Examen de points d'intérêt plus général :
  - protection, fiabilité
  - méthodologie de conception et de réalisation
- Directions de recherches ouvertes.

Rappelons d'abord quelles ont été les idées directrices de notre projet :

1) il s'agissait d'abord de montrer sur un exemple réel comment le processus de conception et de construction d'un système à partage de ressources pouvait être rationalisé, en choisissant au départ un petit nombre de concepts de base et en exploitant de manière aussi complète que possible chacun de ces concepts. C'est une idée qui semble devoir gouverner naturellement toute conception d'un ensemble un peu complexe (c'est en particulier l'un des principes directeurs de la conception du langage ALGOL 68, introduit par ses auteurs [W6] sous le nom d'"orthogonal design"), mais son application aux systèmes d'exploitation est rendue difficile par la complexité des objets réalisés et par la présence de nombreuses contraintes du "monde réel".

2) pour chacun des concepts ainsi introduits, il s'agissait d'examiner sa validité, en ce qui concerne aussi bien son utilisation pour clarifier la conception et la construction d'un système, que sa contribution à la qualité et à

la sécurité du produit final.

3) d'une manière plus précise, nous avons été amenés à développer l'étude des points suivants :

a) utilisation et implantation de la notion de processus et du mécanisme de synchronisation des sémaphores pour la description de toutes les activités du système,

b) mode d'accès unique à l'information en mémoire virtuelle grâce au mécanisme du couplage,

c) utilisation de la notion d'usager comme unité d'allocation de ressources.

4) en ce qui concerne plus spécifiquement l'allocation de ressources, nous avons cherché à évaluer l'efficacité des principes ci-après, suggérés par l'examen de diverses expériences antérieures :

a) considérer l'allocation de toutes les ressources d'un système comme un problème global et non comme une suite de problèmes indépendants,

b) introduire un ordre logique dans les décisions d'allocation ; traduire cet ordre dans un diagramme de transition d'états servant à guider les algorithmes d'allocation,

c) ajuster dynamiquement les décisions d'allocation et libération de ressources à la charge observée du système.

Réservant pour la fin les considérations d'ordre méthodologique, nous allons successivement examiner les points précis énumérés aux 3) et 4) ci-dessus :

- Gestion des activités parallèles

Ce domaine étant traité de façon détaillée dans la thèse de C. KAISER [K3], nous ne lui consacrerons ici qu'une brève discussion. La notion de processus s'est révélée un outil puissant aussi bien pour la description que pour la construction même du système ; son application à la réalisation des mécanismes d'allocation de ressources sera discutée plus loin. L'adoption d'un mécanisme unique de synchronisation constitué par les sémaphores, a permis de représenter clairement les relations entre processus. Avec le recul de l'expérience, on peut maintenant considérer que ce mécanisme se situe à un niveau trop élémentaire : les primitives de synchronisation qu'il comporte étant considérées comme l'équivalent du répertoire des opérations de base d'une machine, il reste à définir des instructions de plus haut niveau. C'est d'ailleurs le sens des recherches menées actuellement par plusieurs auteurs [H2, H3].

- Accès à l'information

Un souci d'uniformité et de simplicité nous a conduit à une représentation unique de l'ensemble de l'information accessible, sous la forme de fichiers de structure linéaire. Le choix de la technique utilisée pour accéder à cette information donne un bon exemple du genre de compromis auquel peuvent conduire les contraintes de réalisation sur un matériel donné (ici l'existence d'une mémoire virtuelle unique, non segmentée). On a prévu la possibilité d'accéder simultanément à travers la mémoire virtuelle à des parties de plusieurs fichiers, sans avoir pour cela à limiter la taille de chacun d'eux. Par ailleurs, on a supprimé les conflits entre mécanisme d'échange (sauvegarde temporaire du contenu d'une mémoire virtuelle sur mémoire secondaire) et mécanisme de transfert mémoire-fichier, grâce au principe de la localisation unique des articles de fichiers. Cette simplification a toutefois sa contre-partie : un transfert d'information entre articles de fichiers se traduit par un transfert d'information en mémoire centrale, puisque les deux articles doivent être couplés à des pages différentes. Une autre difficulté dans l'utilisation du couplage est celle de la protection de l'information ; elle n'est d'ailleurs pas inhérente au mécanisme du couplage proprement dit, mais à l'utilisation de la mémoire virtuelle; on y reviendra à propos de la protection.

Moyennant ces restrictions, la notion de couplage apparaît comme un moyen commode de gérer l'accès à l'information par un ensemble d'usagers à travers une mémoire virtuelle à un niveau. Compte tenu des contraintes dues à l'adressage de la machine (absence de translation), le partage d'informations communes entre usagers par couplage à un fichier unique, ne pose aucun problème supplémentaire.

- La notion d'usager

Essayons à présent de dresser un bilan de l'utilisation de la notion d'usager dans notre système. Rappelons ici qu'un usager, dans le système ESOPE, se caractérise par les propriétés suivantes :

a) un usager est une collection de processus communiquant entre eux mais indépendants (au sens intrinsèque, c'est-à-dire qu'aucune interaction n'est programmée) des autres processus du système ; deux processus d'usagers différents n'ont donc d'interactions qu'à travers les mécanismes d'allocation de ressources.

b) à un usager correspond un domaine de validité de noms de processus et de sémaphores, inaccessibles depuis les autres usagers.

c) à un usager est attachée une mémoire virtuelle unique,

espace d'adressage linéaire. L'accès à l'information contenue dans les fichiers se fait en associant des articles de fichier à des parties de cet espace d'adressage.

d) enfin, l'usager est l'entité à laquelle le système alloue les ressources disponibles ; un usager ne peut d'ailleurs agir que dans une faible mesure (priorités d'exécution) sur la répartition interne de ces ressources entre ses différents processus.

De ces trois propriétés, on peut considérer que a) et b) constituent une définition de l'usager, c) et d) résultant de choix de conception. C'est sur ces choix et sur leurs conséquences qu'il semble intéressant de revenir.

Le choix b) est essentiellement lié à des questions de protection, et sera examiné lors de la discussion correspondante.

En ce qui concerne la propriété c) (mémoire virtuelle attachée à chaque usager et non à chaque processus), un argument déterminant pour le choix effectué est lié à la réalisation : attacher une mémoire virtuelle à chaque processus conduit à multiplier la taille des tables de description de la mémoire virtuelle et implique des temps notables de commutation entre processus d'un même usager, dus au rechargement de la mémoire topographique. Un autre argument était la facilité de communication entre processus d'un même usager via la mémoire virtuelle commune.

Toutefois, le choix effectué comporte un certain nombre d'inconvénients qui ont tous la même origine : en associant plusieurs processus à la même mémoire virtuelle, on a introduit une liaison artificielle entre des objets indépendants. Les conséquences en sont :

- l'absence de protection mutuelle de l'information en mémoire virtuelle entre processus d'un même usager. La protection de la mémoire virtuelle est en effet associée aux pages de cette mémoire même et non aux processus qui y accèdent. En particulier, quand un processus accède au cours de l'exécution de primitives, à des informations de gestion du système lui-même, contenues en mémoire virtuelle (comme les catalogues), ces informations deviennent par là même accessibles sans contrôle aux autres processus de l'usager. La seule protection efficace dans ce cas consiste à imposer l'arrêt de tous les autres processus de l'usager quand un processus exécute une telle primitive.

- l'application à tous les processus d'un usager, sans distinction, des conditions d'exécution définies globalement pour l'usager par des informations en mémoire virtuelle. Citons comme exemple de telles conditions : le traitement non-standard des déroutements, les restrictions sur l'opération de couplage, les droits d'accès aux fichiers.

D'une façon générale, on est ainsi conduit à rendre certains pouvoirs communs à plusieurs processus. Ici encore, si on désire privilégier un processus donné vis à vis de ces pouvoirs, on doit s'arranger pour qu'il s'exécute seul : c'est ce qui est fait dans le cas du processus premier.

Enfin, le choix de considérer l'usager comme unité d'allocation des ressources est une conséquence directe de l'association de la mémoire virtuelle à l'usager, en raison du temps élevé de commutation de mémoire virtuelle par chargement des registres topographiques. La notion d'état des usagers, combinaison des états intrinsèques des processus et de la répartition des ressources a permis de simplifier la conception du système en réduisant le nombre de cas distincts à considérer.

- Allocation de ressources

En ce qui concerne l'allocation des ressources, résumons nos principales décisions et conclusions :

a) nous avons considéré comme un problème global la gestion de l'ensemble des ressources constituées par la mémoire principale, la mémoire secondaire et l'unité centrale. L'unicité du mécanisme d'échange d'information entre mémoire centrale et mémoire secondaire, conséquence du choix du couplage, nous a ici facilité la tâche en réduisant au strict minimum nécessaire le nombre des états différents d'un usager, fonction des ressources allouées ou demandées. La disparité des temps de commutation nous a conduit à allouer systématiquement mémoire et unité centrale dans un ordre fixe. Le graphe des transitions entre états d'un usager pouvait dès lors être construit.

b) la politique de gestion de la mémoire a été guidée par le souci d'éviter un trop grand encombrement, générateur du phénomène d'instabilité connu sous le nom d'écroulement ; toutefois on a veillé à ne pas restreindre inutilement le taux d'utilisation de la mémoire. A cet effet, un mécanisme régulateur a été mis en place, dont l'effet est d'ajuster dynamiquement le degré de multiprogrammation en fonction du comportement des programmes. L'expérience a montré l'efficacité de ce dispositif. Toutefois, si un remède à l'écroulement semble connu, une analyse plus fine de ses causes reste encore à faire ; cette analyse doit nécessairement s'appuyer sur un modèle satisfaisant du comportement des programmes ; plusieurs efforts ont récemment été faits dans ce sens

Nous avons dû,pour des raisons technologiques, nous restreindre à la réalisation d'un mécanisme régulateur assez primitif. L'estimation des besoins futurs d'un programme par l'analyse de son comportement

récent peut se faire de façon plus fine que dans notre réalisation, en utilisant des moyens câblés adéquats. C'est d'ailleurs le sens de l'évolution récente de la technologie, qui grâce à l'utilisation des antémémoires permet de mettre en oeuvre de façon automatique des algorithmes d'allocation fondés sur le comportement des programmes. Mais de tels algorithmes restent encore limités à la prise en compte d'un seul paramètre : la suite des références à la mémoire, estimée de manière plus ou moins fine dans l'espace et dans le temps. Un pas supplémentaire reste encore à franchir : celui de l'utilisation, en temps réel, de mesures plus globales sur le comportement du système pour l'élaboration de ses algorithmes de gestion des ressources. Il serait intéressant que les techniques de régulation utilisées dans d'autres domaines puissent être appliquées aux systèmes d'exploitation ; il ne semble pas que beaucoup d'études aient été faites dans cette voie (voir cependant [S3, W7]).

c) un effort systématique a été fait pour favoriser le partage de l'information, en particulier par l'utilisation de programmes réentrants. Comme cela a été signalé plus haut, le mécanisme d'accès à l'information par couplage se prête bien au partage d'information. On a pu estimer, au moins en simulation, le gain apporté par le partage de l'information. Il reste encore à préciser par des mesures en exploitation réelle les conditions d'utilisation des programmes partagés. Il est important de noter que l'utilisation de programmes partagés n'introduit pas de complications supplémentaires, contrairement à une opinion assez répandue, à condition de le prévoir assez tôt dans la conception.

d) un aspect particulièrement intéressant de la construction de l'allocateur de ressources est sa réalisation au moyen d'une famille de processus parallèles. L'intérêt de la décomposition est multiple :

- mise en évidence des actions logiquement parallèles (c'est-à-dire qui pourraient se dérouler en parallélisme réel sur un multiprocesseur),

- uniformité de la structure du système : l'activité de gestion du système et l'activité des usagers sont représentées par le mécanisme unique des processus ; seuls diffèrent les priorités et les pouvoirs respectifs des deux classes de processus, c'est-à-dire les restrictions apportées à leur jeu d'instructions. En particulier, blocage intrinsèque et blocage technologique sont représentés par le biais d'un même mécanisme et l'attribution de l'unité centrale se fait automatiquement grâce à la structure de la chaîne d'activation unique. De même, le traitement des interruptions est unifié, mettant clairement en évidence leur rôle d'activation sélective d'un processus.

- Fiabilité, protection

Les divers mécanismes de protection utilisés se ramènent à la notion de pouvoir d'un processus ; on entend par là un ensemble d'informations décrivant les actions permises. Toute opération susceptible d'entraîner des actions interdites est assortie d'un contrôle du pouvoir du processus qui l'exécute. Certaines parties du pouvoir d'un processus - et les mécanismes de contrôle associés - sont matérialisés par câblage, comme le mode (maître ou asservi) définissant le jeu d'instructions autorisé, ou l'indicateur d'utilisation de la mémoire topographique, définissant l'information accessible ; d'autres ont dû être programmées, comme l'autorisation d'exécuter certaines primitives du système. La notion d'usager a également fourni un cadre à l'implantation de la protection, en définissant un domaine de validité pour les noms des objets manipulés par les processus (processus ou sémaphores).
La notion de pouvoir n'a pu toutefois recevoir un traitement uniforme, d'une part et surtout en raison de la variété des mécanismes (câblés ou programmés) mis en jeu, d'autre part parce qu'elle ne nous est apparue en fait que progressivement au cours de la conception du système.

- Méthodologie

Dans l'état actuel de nos connaissances, une discussion sur la méthodologie de conception et de programmation de systèmes informatiques est une dangereuse navigation entre deux écueils : l'énoncé de principes généraux, à peu près évidents mais difficiles à appliquer en raison même de leur généralité, et la description de réalisations particulières difficilement transposables à un autre contexte.

Par ailleurs, il serait inexact d'affirmer que les règles que nous allons énoncer nous ont guidé depuis l'origine du projet. Leur utilité s'est plutôt manifestée progressivement, souvent à l'occasion d'erreurs.

- "Principe d'indépendance" : ne pas introduire de liaisons artificielles entre des objets indépendants. Nous avons vu que cette règle n'avait pas été respectée dans le cas des processus d'un même usager, entre lesquels on a introduit des liaisons parasites provenant du partage d'une mémoire virtuelle. Par ailleurs, ce principe nous a été utile pour analyser le fonctionnement du système sous forme d'une famille de processus parallèles.
- "Principe d'uniformité" : utiliser les mêmes mécanismes pour exécuter des actions de même nature dans des circonstances différentes. Ce principe a des incidences au niveau de la conception (par exemple choix d'un mécanisme unique pour tous les transferts entre mémoire principale et mémoire secondaire)

comme au niveau de la réalisation (essayer de rendre les algorithmes du système aussi paramétrés que possible, c'est-à-dire utiliser une procédure unique avec des données variables pour exécuter une classe d'opérations reconnues comme étant de même nature). Une violation manifeste de ce principe est l'absence d'une procédure uniforme traitant les entrées-sorties sur fichiers et sur terminaux ; cela est dû à ce que le problème des entrées sorties n'a été considéré qu'assez tardivement dans la conception du système, une fois prises des décisions irréversibles sur la communication avec les terminaux.

- "Principe du report des décisions" : reporter les décisions les plus contraignantes jusqu'au moment le plus tardif où elles peuvent être prises. Ce principe est valable lors de la phase de conception mais aussi dans le fonctionnement même du système. L'idée directrice est ici que l'on disposera d'une plus grande quantité d'informations et que l'on respectera la liberté de décision le plus longtemps possible. Une application de ce principe peut être trouvée dans la gestion des pages physiques.

En ce qui concerne la réalisation, deux points peuvent être mis en relief : l'utilisation d'un langage de plus haut niveau qu'un assembleur, et la méthode du développement des versions successives du système. Les avantages apportés par l'utilisation du langage LP 10070 ont été ceux attendus, c'est-à-dire la facilité de programmation, et de modification, la clarté et la communicabilité du programme produit ; les réserves que l'on pourrait faire concernent surtout les structures de données, trop rudimentaires, et l'absence de gestion automatique des registres, c'est-à-dire précisément des caractéristiques qui rapprochent le LP 10070 d'un langage d'assemblage.

La réalisation des versions successives du système a été guidée par la décomposition en processus. Nous avons pris le parti d'introduire dès les premières versions l'ensemble des processus constituant le moniteur, de manière à ce que la validité de la décomposition et du mode de communication puisse être expérimentée le plus tôt possible. On a ainsi essayé de séparer les problèmes de synchronisation entre processus des problèmes du développement des algorithmes proprement dits de chaque processus. L'expérience a montré le bien-fondé de cette démarche.

- Voies de recherches ouvertes

Au cours de la discussion qui précède, nous avons déjà évoqué divers prolongements possibles des études auxquelles a donné lieu notre travail. Nous dresserons ici un rapide inventaire des voies qui nous semblent intéressantes ; nous nous limitons ici au seul domaine de l'allocation de ressources.

- développement de programmes de mesures permettant une meilleure compréhension du fonctionnement interne des systèmes. Le but recherché est de pouvoir formuler avec plus de sûreté les hypothèses initiales nécessaires à la conception de tout système.

- élaboration de modèles de comportement des programmes, dans le but d'étudier de façon plus approfondies les questions suivantes :
  - "écroulement" des systèmes et ses remèdes
  - utilisation des programmes partagés

- mise en oeuvre d'algorithmes d'allocation de ressources fondés sur des mesures du comportement du système ; étude de la stabilité de tels algorithmes ; corrélativement, détermination des paramètres caractérisant la charge d'un système, c'est-à-dire le degré d'utilisation de ses ressources, et essai de définition de critères d'optimalité.

On peut noter pour conclure que l'évolution de la technologie, en particulier la tendance vers le développement de mémoires non rotatives et de grande capacité, à des prix plus bas, modifiera sans doute profondément l'aspect des problèmes d'allocation de ressources en changeant les ordres de grandeur relatifs des divers facteurs en jeu. Il ne semble pas toutefois que doive être remis en question l'intérêt des voies d'approche qui viennent d'être proposées.

BIBLIOGRAPHIE

Abréviations

AFCET : Association Française pour la Cybernétique Economique et Technique
ACM : Association for Computing Machinery
CACM : Communications of the ACM
JACM : Journal of the ACM
AFIPS : American Federation of Information Processing Societies
FJCC : AFIPS Fall Joint Computer Conference
SJCC : AFIPS Spring Joint Computer Conference
IFIP : International Federation for Information Processing
OSPk : k^{th} ACM Symposium on Operating Systems Principles
OSP1 : Gatlinburg, Tenn, (oct 1967)
OSP2 : Princeton, N.J., (oct 1969)
OSP3 : Stanford, Cal, (oct 1971)
OST : International Seminar on Operating Systems Techniques, Belfast (sept. 1971)

(A1) ALDERSON,A. ; LYNCH,W.C. ; RANDELL,B.
Thrashing in a Multiprogrammed Paged System.
OST (1971)

(A2) ANDERSON, G.B. et al.
Design of a Time-sharing System Allowing Interactive Graphics
Proc. ACM. 23rd Nat. Conf (1968)

(B1) BENSOUSSAN,A. ; CLINGEN,C.T. ; DALEY,R.C.
The Multics Virtual Memory
OSP2 (1969)

(B2) BETOURNE,C. ; BOULENGER,J. ; FERRIE,J. ; KAISER,C. ;
KRAKOWIAK,S. ; MOSSIERE,J.
Process Management and Resource Sharing in the Multiaccess System ESOPE.
OSP2 (1969), and CACM, 13, 12 (dec 1970)

(B3) BETOURNE,C. ; FERRIE,J. ; KAISER,C. ; KRAKOWIAK,S. ; MOSSIERE,J.
System Design and Implementation using Parallel Processes
Proc IFIP Conference (Ljubljana, 1971)

(B4) BETOURNE,C. ; BOULENGER,J. ; FERRIE,J. ; KAISER,C. ;
KRAKOWIAK,S. ; MOSSIERE,J.
Le Système ESOPE :
1ère partie : Présentation Générale
2ème partie : Notion d'espace virtuel
3ème partie : Allocation de ressources
Congrès AFCET (Paris, 1970)

(B5) BETOURNE,C. ; KRAKOWIAK,S.
Simulation de l'allocation de ressources dans un système conversationnel à mémoire virtuelle paginée.

(B6) BRAWN,B.S. ; GUSTAVSON,F.G.
Program Behavior in a Paging Environment
Proc FJCC (1968)

(C1) COFFMAN,E.G.jr ; ELPHICK,M.J. ; SHOSHANI,A.
System Deadlocks
Computing Surveys, 3,2 (June 1971)

(C2) COFFMAN,E.G., jr ; RYAN,T.A.
A Study of Storage Partitioning using a Mathematical Model of Locality
OSP3 (1971)

(C3) COFFMAN,E.G. jr ; VARIAN,L.C.
Further Experimental Data on the Behavior of Programs in a Paging Environment.
CACM, 11,7 (July 1968)

(D1) DALEY,R.C. ; NEUMANN,P.G.
A General-purpose File System for Secondary Storage
Proc FJCC (1964)

(D2) DENNING,P.J.
Resource Allocation in Multiprocess Computer Systems
MAC TR-50 (Ph. D. Thesis) MIT (1968)

(D3) DENNING,P.J.
Thrashing : Its causes and prevention
Proc. FJCC (1968)

(D4) DENNING,P.J. ; SCHWARTZ,S.C.
Properties of the Working Set Model
OSP3 (1971)

(D5) DIJKSTRA,E.W.
Cooperating Sequential Processes
in "Programming Langages", F. GENUYS, editor, NATO Adv. Studies Institute. Academic Press (1968)

(D6) DIJKSTRA,E.W.
The Structure of the "THE" Multiprogramming System
OSP1 (1971), and CACM, 11,5 (May 1968)

(F1) FERRIE,J.
Gestion des Processus dans un système à Partage de Ressources
Thèse de Docteur-Ingénieur, Toulouse (1971)
reproduit dans les cahiers de l'IRIA, n° 8, (1972)

(F2) FINE,G.H. et al.
Dynamic Program Behavior under Paging
Proc. ACM Nat. Conf. (1966)

(H1) HABERMANN,A.N.
Prevention of System Deadlocks
CACM, 12,7 (July 1969)

(H2) HANSEN,P.B.
Short-term Scheduling in Multiprogramming Systems.
OSP3(1971)

(H3) HOARE,C.A.R.
Towards a Theory of Parallel Programming
OST (1971)

(H4) HOLT,R.C.
On Deadlocks in Computer Systems
Ph.D. Thesis, Cornell University (1971)

(K1) KEMENY,K.G ; SNELL,J.L.
Finite Markov Chains
Van Nostrand (1960)

(K2) KUEHNER,C.J ; RANDELL,B.
Demand Paging in Perspective
Proc. FJCC (1968)

(K3) KAISER, C.
Conception et réalisation de systèmes à accès multiple - Gestion du parallélisme
Thèse d'Etat (à paraître)

(K4) KING, W.F.
Analysis of demand paging algorithms
Proc. IFIP Conference (Ljubljana, 1971)

(L1) LAMPSON,B.W.
A Scheduling Philosophy for Multiprocessing Systems.
OSP1 (1967) and CACM, 11,5 (May 1968)

(M1) De MEIS, M.M. ; WEIZER, N.
Measurement and Analysis of a Demand Paging Time-sharing System.
Proc ACM. Nat. Conf. (1969)

(M2) MOSSIERE, J.
Allocation de Ressources dans un Système à Accès Multiple
Thèse de Doctorat 3è cycle, (Paris 1971)
reproduit dans les cahiers de l'IRIA, n° 8, (1972)

(P1) PINKERTON, T.B.
Program Behavior and Control in Virtual Storage Computer Systems. Ph. D. Thesis, University of Michigan (1968)

(R1) RANDELL,B. ; KUEHNER,C.J.
Dynamic Storage Allocation Systems
Comm. ACM, 11,5 (May 1968)

(S1) SALTZER,J.H.
Traffic Control in a Multiplexed Computer System
MAC-TR-30 (Ph.D. Thesis) MIT (1966)

(S2) SCHERR,A.L.
An Analysis of Time-Shared Computer Systems.
MAC-TR-18 (Ph.D.Thesis) MIT (1965)

(S3) SHILS,A.J.
The Load Leveler
IBM Research Report RC 2233 (1968)

(S4) Software Engineering Techniques (NATO Conference)
P.Naur, B.Randell, ed.
(Garmisch 1968)

(S5) Software Engineering Techniques (NATO Conference)
J.N. Buxton, B. Randell, ed.
(Rome 1969)

(W1) WATSON,R.W.
Timesharing System Design Concepts
Mc Graw-Hill (1970)

(W2) WINOGRAD,J. ; MORGANSTEIN,S.J. ; HERMAN,R.
Simulation of a Virtual Memory, Time-Shared, Demand Paging Operating System
OSP 3 (1971)

(W3) WIRTH,N.
PL 360, A Programming Language for 360 Computers
JACM, 15, 1 (Jan 1968)

(W4) WIRTH, N.
On Multiprogramming, Machine Coding, and Computer Organization.
CACM, 12,9 (Sept 1969)

(W5) WHITFIELD, H.
The organization of the University of Edinburgh Time-Sharing System.
International Seminar on Advanced Programming System, 11,5
Jerusalem (1968)

(W6) Van WIJNGARDEN, A. (Editor); MAILLOUX, B.J. ; PECK, J.E.L ; KOSTER, C.H.A.
Report on the Algorithmic Language Algol 68
Mathematisch Centrum, MR101, Amsterdam (1969)

(W7) WILKES, M.V.
Automatic load adjustment in Time-sharing Systems.
ACM Workshop on Computer Performance Evaluation
Harvard (1971)

Annexe

Notations

On précise dans cette annexe les notations utilisées dans la description des divers algorithmes.

1. Langage

On a utilisé un langage voisin d'ALGOL 60 ; les différences principales sont les suivantes :

1.1 Variables : pour alléger l'écriture, on a omis les déclarations de variables, qui sont reportées dans la présente annexe (§ 2). Les variables utilisées dans le système sont des éléments de tables : ce sont en général des tableaux d'entiers à un indice. Certaines tables existent à plusieurs exemplaires (par exemple une TPV par usager) ; la table utilisée est alors précisée en complétant son nom par celui d'un désignateur, les deux noms étant séparés par un point (ex. : TPV.XUS [V]).

1.2 Opérateurs : la concaténation de deux listes chaînées est représentée par l'opérateur ||

1.3 Instructions : on a introduit les instructions suivantes :

while <condition> do <instruction> ;

repeat <instruction> until <condition> ;

Leur signification se passe de commentaire. L'instruction for a été étendue comme suit :

for (<variable> ∈ <tableau>) & (condition) do <instruction>;

La variable contrôlée prenant successivement toutes les valeurs des éléments du tableau, l'<instruction> est exécutée si la <condition> est vraie.

2. Variables d'état du système

2.1 Tables associées à l'usager

2.11 Table des usagers TUS : - Nombre d'entrées NUS
- entrée : n° d'usager

SUCUS	liens de chaînage vers le successeur	(octet)
PREDUS	et les prédécesseur dans la file à laquelle appartient l'usager	(octet)

ETAT	Etat de l'usager 'E' pour éligible 'S' pour suspendu 'O' pour oisif 'P' pour prêt	(octet)
PRIOUS	priorité de l'usager	(octet)
XCAT	numéro de catégorie courante	(octet)
PTTPD	pointeur vers la Table des Pages Virtuelles	(mot)
PTTFU	pointeur vers la Table des Fichiers de l'usager	(mot)
HEJECT	heure du départ de l'usager de la mémoire	(mot)

2.12 Table des Pages Virtuelles (TPV) :

Cette table définit à chaque instant la composition de la mémoire virtuelle d'un usager.

- on y accède par l'intermédiaire du champ PTTPV de la TUS,
- 256 entrées,
- index d'entrée : numéro de page virtuelle.

FTPV : numéro de fichier, fixé au couplage et remis à zéro à la fermeture du fichier
(FTPV = 0 correspond à une page virtuelle non couplée)
(demi-mot).

ATPV : numéro d'article de fichier, fixé au couplage
(demi-mot).

PTPV : numéro de page physique, initialement à zéro, mis à jour à l'acquittement de la requête, remis à 0 lorsque le fichier est fermé ou la page recouplée. (octet).
L'ensemble des champs PTPV est utilisé pour le chargement de la mémoire topographique.

PE : protection effective de la page initialement à '11', évolue en même temps que l'état de la page physique, remise à '11' au départ de l'usager de l'état courant.
(2 bits).
L'ensemble des champs PE est utilisé pour le chargement des registres de protection virtuelle.

PF : protection affectée au fichier associé à la page
(2 bits).

2.13 Table des Pages Demandées (TPD) :

On note dans cette table, en vue de la préparation et de l'acquittement des requêtes, tous les renseignements nécessaires sur les pages demandées par l'usager.

- on y accède par l'intermédiaire du champ PTTPD de la TUS,
- LTPD entrées,
- accès séquentiel.

VTPD numéro de page virtuelle (octet)

FTPD, ATPD : coordonnées de l'article demandé (demi-mot)

PTPD numéro de page physique fixée à la préparation de la requête (octet)

PFTPD : protection du fichier FTPD

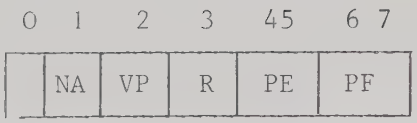
PETPD : protection effective à placer dans la TPV à l'acquittement de la requête, calculée en fonction de la protection actuelle, de PFTPD et du type d'accès.
PETPD = '11' si la demande est abandonnée au cours de la préparation de la requête (page demandée par un autre usager, mais encore en transit).

R nature de la tentative d'accès à l'article (R = 1 si écriture).

VP bit égal à 1 si la page allouée à VTPD doit être vidée.

NA bit égal à 1 dans le cas de la création d'un nouvel article.

Les cinq dernières informations sont regroupées dans un octet de format suivant :



2.14 Table des Fichiers Utilisés par l'usager (TFU)

Cette table est utilisée lors du départ de l'usager de l'état courant, pour libérer les pages allouées à cet usager.

- on y accède par l'intermédiaire du champ PTTFU de la TUS,
- 32 entrées, les 8 premières réservées à des fichiers partagés,
- chaque entrée occupe un demi-mot et contient un numéro de fichier.

2.15 Table des catégories :

Cette table est définie une fois pour toutes à l'initialisation du système ; son contenu devra être ajusté après expériences.

- nombre d'entrées : NCAT,
- index d'entrées numéro de catégorie

CATINF : numéro de la catégorie inférieure (octet)
CATSUP : numéro de la catégorie supérieure (octet)
VOLMC : taille de mémoire à réserver, exprimée en nombre de pages (octet)
TMC : temps de résidence en mémoire centrale, exprimé en 1/8 de msec. (demi-mot)

2.2 Tables associées aux fichiers :

Nous nous limitons ici aux champs utilisés pour l'allocation de mémoire centrale.

2.21 Table des Fichiers Opérationnels (TFO)

- LTFO entrées
- index d'entrée : numéro de fichier.

On affecte à chaque fichier interne lors de sa création, ou de son passage à l'état interne, un numéro interne f, numéro qui est rendu disponible dès que le fichier est détruit ou devient externe.
On opère de la façon suivante :
on entretient un pointeur vers une entrée libre de la TFO ; à

chaque allocation d'entrée, ce pointeur est mis à jour et pointe sur l'entrée libre suivante, le pointeur balayant la table circulairement.
Comme l'entrée 0 n'est pas utilisée (un numéro de fichier égal à 0 indique en particulier qu'une page virtuelle n'est couplée à aucun article), ce pointeur a la valeur 0 lorsque la TFO est saturée.

- PTTIF : pointeur vers la Table d'Implantation du fichier (mot)
- NUSL : nombre d'usagers ayant ouvert le fichier ; augmente de 1 à toute fermeture. (octet)
- NUSC : nombre d'usagers courants partageant le fichier, augmente de 1 lorsqu'un usager utilisant f entre en mémoire ; diminue de 1 lorsqu'un usager utilisant f quitte la mémoire. (octet)
- NPF : nombre total de pages physiques contenant un article de f ; augmente de 1 à chaque allocation de page au fichier, diminue de 1 à chaque réallocation de page contenant un article de f. (octet)
- NPI, QPI : nombre de pages et page de queue de la liste des pages intactes du fichier, ces deux champs étant regroupés dans un mot.
- NPE, QPE : nombre de pages et page de queue de la liste des pages modifiées du fichier.
- NPMXF : taille maximale du fichier (demi-mot)
- LTIF : longueur courante de la Table d'Implantation du fichier (demi-mot).

Lorsque le fichier n'est plus ouvert par aucun usager et n'a plus d'article en mémoire centrale, on remet à zéro le champ PTTIF, qui peut ainsi servir au repérage des entrées disponibles de la TFO.

2.22 Table d'Implantation de Fichier (TIF)

On associe une TIF à tout fichier interne.

- on accède à la TIF par l'intermédiaire du champ PTTIF de la TFO, sa longueur LTIF figurant également dans cette table,

- index d'entrée : numéro d'article dans le fichier,

- la longueur de la TIF peut varier dynamiquement à la création de nouveaux articles.

- PTIF : numéro de page physique (octet)
  Si ce numéro est différent de 0, la page physique PTIF est allouée à l'article ; il faut se reporter à la table des pages physiques pour savoir si l'article est en cours de chargement ou effectivement utilisable.
  Dès que la page est réallouée à un autre article, PTIF est remis à zéro.

- DTIF : adresse en mémoire secondaire (mot)
  si DTIF est différent de zéro, la page de disque correspondante est destinée à contenir l'article.

- U : ce bit, initialement à 0, est mis à 1 dès la création effective de l'article (première écriture dans la page virtuelle couplée à l'article).

Remarque : rôle du bit U.

L'espace sur disque est alloué à un fichier par ensemble de plusieurs pages de disques consécutives, ensemble appelé granule.
Une page de disque peut dès lors être affectée à un article, (DTIF ≠ 0) avant que l'article ne soit créé ; le bit U permet de détecter des tentatives d'accès en lecture ou en exécution dans un article non encore créé.

2.23 Table des Pages Physiques (TPP)

- 256 entrées
- index d'entrée : numéro de page physique,
  - FTPP : numéro de fichier (demi-mot)
  - ATPP : numéro d'article (demi-mot)

Pour une page non en transit, ces champs fournissent les coordonnées de l'article contenu dans la page ; pour une page en transit, on ne les met à jour qu'à la fin de l'éventuel vidage du contenu de la page, bien qu'on demande simultanément

au module de gestion des disques le vidage et le chargement du nouvel article.

- SUCPP : successeur dans la liste de pages (octet)

- PREDPP : prédécesseur dans la liste de page (octet)

Ces champs sont mis à jour à chaque transfert de la page d'une liste à une autre.

- A : bit d'allocation mis à 1 dès que la page est allouée à un usager courant, remis à 0 à la libération de la page.

- L : bit d'écriture mis à 1 :
  - soit après un déroutement pour tentative d'écriture dans une page allouée intacte ; cette mise à 1 s'accompagne d'un changement de liste de la page,
  - soit au cours d'une préparation de requête, lorsqu'on alloue une page à un article accédé en écriture.

  Ce bit est remis à 0 après un vidage préventif du contenu de la page ; il est réassigné à toute nouvelle allocation.

- T : bit de transit ; mis à 1 :
  - soit au début d'un vidage préventif,
  - soit à l'allocation de la page, lorsque l'allocation s'accompagne d'une demande de chargement, éventuellement précédé d'un vidage.

  mis à 0 à l'acquittement de la requête.